DECENTRALIZED ACCESS CONTROL SYSTEM
Models for Analysis and Enforcement

A Thesis

Submitted For the Degree of

Master of Science (Engineering)

in the Faculty of Engineering

by

Arul Ganesh

Computer Science and Automation

Indian Institute of Science

BANGALORE – 560 012

MARCH 2008

TO

The GOD almighty

Acknowledgements

I would like to express my deep and sincere gratitude to my supervisor, Prof. K. Gopinath, Department of Computer Science and Automation, Indian Institute of Science (IISc). His wide knowledge and logical way of thinking have been of great value for me. His understanding, encouraging and personal guidance have provided a good basis for my thesis. Many a times he has provided new insights and problem definitions which ensured a change of course of my work towards better results. It was a great pleasure for me to conduct this research under his supervision. I warmly thank my supervisor Dr. Suman Roy for his valuable advice and friendly help. I wish to thank Mr. Yalla Veera Prakash, Mr. Mahalingam Vasudevan, Mr. Srinivas Raghavan, and Dr. Krishna Mikkilineni of Honeywell Technology Solutions Lab for permitting me to register for this program. I owe my loving thanks to my family. They have lost a lot of my personal time due to my research as an external registrant. Without their encouragement and understanding it would have been impossible for me to finish this work. My special thanks are due to my friends and colleagues at Honeywell Technology Solutions Lab for their support especially the members of my team in 'Communication and Computation Systems' group who are a pleasure to work with.

Publications based on this Thesis

1. "SPKI/SDSI Certificate Chain Discovery with Generic Constraints", ACM Compute 2006.
2. "A Framework for Decentralized Access Control", 2007 ACM Symposium on InformAtion, Computer and Communications Security (ASIACCS'07).

Abstract

The SPKI/SDSI is a security infrastructure whose principal goal is to facilitate the building of secure, scalable, distributed computing systems. In SPKI/SDSI the principals are the public keys and each public key is a certificate authority. Given a set of SPKI/SDSI certificate, the decision on granting access to a resource by a user is taken by using a certificate-chain discovery process. A certificate-chain for a set of certificates $\mathcal{C}$ is of the form $c_k \circ c_{k-1} \circ \cdots \circ c_1$, where c_1, c_2, ..., c_k are certificates in $\mathcal{C}$. Given the set of certificates $\mathcal{C}$, the public-key of resource owner K_r, client requesting access to the resource K_c, and access right T requested, a certificate-chain discovery algorithm looks for a finite set of certificate chains proving that K_c is allowed access specified by T on the resource K_r. As per RFC2693 SPKI/SDSI infrastructure allows validity specification. The validity specification is a time period during which a certificate is valid, assuming the signature verifies. The certificate expired beyond the validity period specified against the certificate. The validity specification takes the form (t_1, t_2), specifying that the certificate is valid from time t_1 to time t_2, both inclusive. This validity specification, as defined in the RFC-2693, allows for a limited constraints on the certificate. The RFC-2693 also allows for more powerful constraints specification that can be associated to the certificates of the SPKI/SDSI system. In this report it is demonstrated how Weak monadic Second-order theory of 1 Successor (WS1S) can be used for specification of more generic validity constraints. It is also shown that the WS1S logic can be combined with Weighted Pushdown System (WPDS) to formally specify more generic constraints and answer authorization questions that are based on the constraints. This is accomplished by augmenting the WPDS, called AWPDS, so that the rules of WPDS are associated with WS1S formula and the transitions of the prefix closed configuration automaton generated from WPDS is annotated with regular language, which is succinctly represented by an automaton. The concepts described has been implemented as a tool called SCAT and the feasibility of using WS1S for the description of complex constraints has been demonstrated by implementing a parser-complier named FACT, which uses a physical access control domain specific language and outputs WS1S formula.

Contents

List of Tables

List of Figures

Keywords

Access Control, SPKI/SDSI, Authorization, Pushdown System, Weighted Pushdown System.

Notation and Abbreviations

Terms

Subject – The entity of an access control system that accesses resources like files, directories.

Object – The resource in a system that is protected by the access control system.

Access Control – The means of controlling access to an object by a subject.

Policy – A statement of what is, and what is not, allowed in an access control system.

Authentication – A method of verifying the identity of a user or an entity in an access control system.

Authorization – The process of determining what access rights are provided to the subject over an object.

Access Control List – A list that specifies what access privileges subjects have over an object.

Issuer – The public-key that signs the certificate.

Validity Specification – A time period during which a certificate is valid, assuming the signature verifies. Beyond this period, the certificate has expired, and should be renewed.

Identifier – A word over some alphabet used in SPKI/SDSI system.

Tag – The specific authorization or authorizations being granted from the issuer to the subject.

Delegation Bit – A boolean value which if set to true for a certificate specifies that the subject of this certificate is able to grant to other principals any subset of the authorization that it is receiving from the issuer of this certificate. If it is set to false for the certificate it specifies that the issuer is not delegating to the subject the authority granted to it.

Certificate – A digital signature to bind together a public key with an identity information such as the name of a person or an organization, their address, and so forth. The certificate can be used to verify that a public key belongs to an individual.

Threshold Subjects – The minimum number of subjects from a given set of subjects who must agree to sign the request before authorization can be propagated through the certificate.

Lattice – A partially ordered set (or poset), in which all nonempty finite subsets have both a supremum (join) and an infimum (meet).

Principal – An entity in access control system that can be positively identified and verified using authentication techniques.

Confidentiality – The process of concealment of information or resources.

Integrity – The trustworthiness of data or resources. It is usually phrased in terms of preventing improper or unauthorized change.

Availability – The ability to use the information or resource desired at the moment of request.

Threat – A potential violation of security.

Attack – The actions that causes security threats to be realized.

Disclosure – An unauthorized access to information.

Deception – An act resulting in an acceptance of false data.

Disruption – An interruption or prevention of correct operation.

Usurpation – An unauthorized control of some part of a system.

Detection – The mechanisms that aids in reporting that the data's integrity is no longer trustworthy.

Secure System – A system that starts in an authorized state and cannot enter an unauthorized state.

Breach Of Security – The act of the system being in an unauthorized state.

Integrity Policy – The parts of the security policy that describes the conditions and manner in which data can be altered.

Security Model – An abstract model of security used to understand the security system.

Discretionary Access Control – An access control mechanism in which an individual user

can set an access control mechanism to allow or deny access to an object.

Mandatory Access Control – An access control mechanism in which the system controls access to an object and an individual user cannot alter that access.

Originator Controlled Access Control – An access control system that bases access decision on the creator of an object or the information it contains.

Granting (Authorization) – The act of providing access permission that was provided to a subject over an object or set of objects.

Revoking (Authorization) – The act of removing access permission that was provided to a subject over an object or set of objects.

User – The subjects in the access control system who access the objects.

Client – The computer system or the user which is the subject that access the object in access control system.

Context Based Access Control – An access control system that is characterized by a context that defines the state of a system at a given time.

Group – A set of principals in SPKI/SDSI.

Certificate Chain – A base certificate plus a sequence of all certificates which establishes connection between the certificates.

Authorization Flow – The flow of authorization from one principal to another using delegation in SPKI/SDSI.

Symbols

$\mathcal{K}$ – The set of public keys in SPKI/SDSI.

K, K_A, K_B, K' – The specific keys in SPKI/SDSI.

Σ – The alphabet from which the set of words is formed which form the subset of an *identifier* in SPKI/SDSI system.

$\mathcal{A}$ – The set of identifiers in SPKI/SDSI.

Term – A key follow by 0 or more identifiers. Terms are either keys, local names, or extended names.

Local Name – The name in SPKI/SDSI which is of the form K `A`, where $K \in \mathcal{K}$ and $\texttt{A} \in \mathcal{A}$ is an identifier.

Extended Name – The name in SPKI/SDSI of the form $K\ \sigma$, where $K \in \mathcal{K}$ and σ is a sequence of identifiers of length greater than one.

$\mathcal{T}$ – The set of terms in SPKI/SDSI certificate system.

$\mathcal{C}$ – A finite set of SPKI/SDSI certificates.

$\mathcal{K}_{\mathcal{C}}$ – Keys that appear in $\mathcal{C}$.

$\mathcal{I}_{\mathcal{C}}$ – Identifiers that appear in $\mathcal{C}$.

$\mathcal{W}$ – Weighted Pushdown System.

$\mathcal{A}$ – Augmented Weighted Pushdown System.

Acronyms

ACF – Access Control Function.

ACM – Access Control Matrix.

AWPDS – Augmented Weighted Pushdown System.

MAC – Mandatory Access Control.

DAC – Discretionary Access Control.

ACL – Access Control List.

SPKI/SDSI – Simple Public Key Infrastructure and Simple Distributed Security Infrastructure.

WPDS – Weighted Pushdown System.

PDS – Pushdown System.

GPP – Generalized Pushdown Predecessor.

GPR – Generalized Pushdown Reachability.

GPS – Generalized Pushdown Successor.

FSM – Finite State Machine.

CRL – Certificate Revocation List.

PKI – Public Key Infrastructure.

IBAC – Identity Based Access Control.

RBAC – Role Based Access Control.

Chapter 1

Introduction

1.1 Access Control – An Overview

Security of a resource (physical or logical) rests on three fundamental security services which are – confidentiality, integrity, and availability. The interpretations of these three aspects vary, as do the contexts in which they arise. One way to ensure security of a resource is to provide managed access to the resource based on a set of permissions or policies that are defined for the resource and the agent accessing the resource. The means of managing access to resource is called *Access Control*. An access control system consists of policies, which protect resources, which is generically referred to as *Object* by agents that are interested in accessing the resources, which is referred to as *Subject*.

Confidentiality is concerned with the concealment of information or resources. Access control mechanisms support confidentiality by guarding access to resources based on *Policy Specification* and its enforcement. A policy specification unambiguously specifies the type of access that a set of Subjects have over a set of Objects.

Integrity refers to the trustworthiness of data or resources, and it is usually phrased in terms of preventing improper or unauthorized change. Integrity includes data integrity and origin integrity. Integrity mechanisms fall into two classes: *prevention* mechanisms and *detection* mechanisms. Prevention mechanisms seek to maintain the integrity of the data by blocking any unauthorized attempts to change the data or any attempts to change the data in unauthorized ways. The former occurs when a user authorized to make certain changes in the data tries to change the data in other ways. Detection mechanisms do not try to prevent violations of integrity; they simply report that the data's integrity is no longer trustworthy. Detection mechanism may analyze system events to detect problems or may analyze the data itself to see if required or expected constraints still hold. Working with integrity is very different from working with confidentiality. With confidentiality, the data is either compromised

or it is not, but integrity includes both the correctness and the trustworthiness of the data. Integrity in a system that changes with the context is a challenge. A security system that that evolves with the changing context is required.

Availability refers to the ability to use the information or resource desired. Availability is an important aspect of reliability as well as of system design because an unavailable system is at least as bad as no system at all. A major factor influencing availability is the ability of the system to tolerate faults. The factor of availability can be built into a access control system during the policy specification stage. This also means that degree of availability of a system can be measured given the specification of access control.

Threat is a potential violation of security. The violation need not actually occur for there to be a threat. The fact that the violation might occur means that those actions that could cause it to occur must be guarded against. Those actions are called *Attacks*. The three security services – confidentiality, integrity, and availability – counter threats to the security systems. Threats can be divided into four broad classes: *disclosure*, or unauthorized access to information; *deception*, or acceptance of false data; *disruption*, or interruption or prevention of correct operation; and *usurpation*, or unauthorized control of some part of a system. Analysis of security system is an important method to ensure that threats are avoided and services are unhindered.

In any access control system there exists a crucial difference between the policy and the mechanism.

Definition 1. A *Security Policy* is a statement of what is, and what is not, allowed.

Definition 2. A *Security Mechanism* is a method, tool, or procedure for enforcing a security policy.

Policies may be presented mathematically, as a list of allowed (secure) and disallowed (non-secure) states. Given a security policy's specification of "secure" and "non-secure" actions the security mechanisms can prevent the attack, detect the attack, or recover from the attack. The strategies may be used together or separately. *Prevention* means that an attack will fail. *Detection* is most useful when an attack cannot be prevented, but it can also indicate the effectiveness of preventive measures. *Recovery* attempts to stop an attack and later to assess and repair any damage caused by that attack.

The *design* of a system translates the specifications into components that ensures that system is correctly partitioned into secure and non-secure states. Given a design, the implementation creates a system that satisfies the design. If the design also satisfies the specification, then by transitivity the implementation will also satisfy the specifications. The cycle of identification of threat followed by design and operation of security mechanism is shown in Figure 1.1.

1.1.1 Security Policies

The Definition 1 is formalized here by using the concepts of secure and non-secure states introduced in the previous section. This definition provides an alternative view of the security policies.

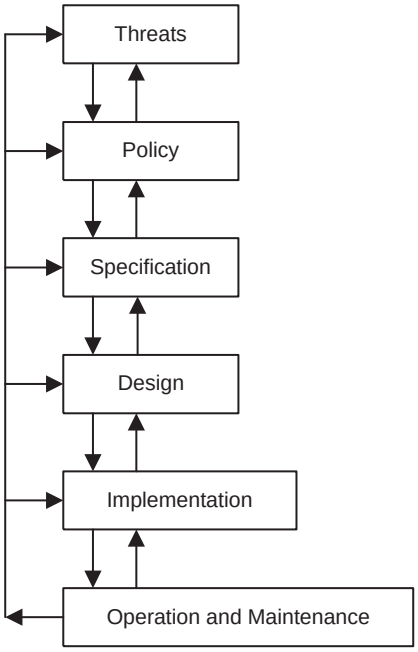

Figure 1.1: Security Cycle

Definition 3. A *Security Policy* is a statement that partitions the states of the system into a set of *authorized*, or *secure*, states and a set of *unauthorized*, or *non-secure*, states.

The definition of three basic security services are formalized below in the context of the formal definitions provided in this section. A security policy sets the context in which a secure system can be defined. It may be noted that what is considered secure under one policy may not be secure under a different policy.

Definition 4. A *Secure System* is a system that starts in an authorized state and cannot enter an unauthorized state.

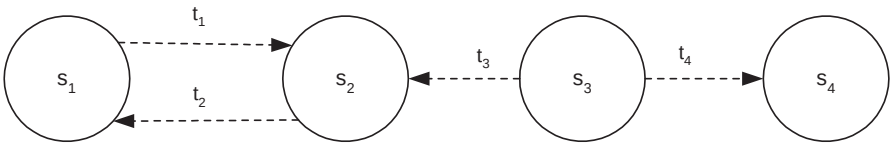

Figure 1.2: A finite-state machine for a secure system in which the authorized states are s_1 and s_2.

A security policy considers all relevant aspects of confidentiality, integrity, and availability. With respect to confidentiality, it identifies those states in which information leaks to those not authorized to receive it. This includes not only leakage of rights but also the illicit transmission of information without leakage of rights, called *information flow*. Also, the policy must handle dynamic changes of authorization, so it includes a temporal element. This aspect of policy is called *confidentiality policy*.

With respect to integrity, a security policy identifies authorized ways in which information may be altered and entities authorized to alter it. Authorization may derive from a variety of relationships, and external influences may constrain it. Those parts of the security policy that describes the conditions and manner in which data can be altered are called the *integrity policy*. With respect to availability, a security policy describes what services must be provided.

Definition 5. A *security model* is a model that represents a particular policy or set of policies.

A model abstracts details relevant for analysis. Analysis rarely discuss particular policies; and focus on *specific characteristics* of policies, because many policies exhibit these characteristics; and the more policies exhibit those characteristics, the more useful the analysis.

A security policy may use two types of access controls, alone or in combination. In one, access control is left to the discretion of the owner. In the other, the system controls access, and the owner cannot override the controls.

Definition 6. If an individual user can set an access control mechanism to allow or deny access to an object, that mechanism is a *discretionary access control (DAC)*, also called an *identity-based access control (IBAC)*.

Discretionary access controls base access rights on the identity of the subject and the identity of the object involved. Identity is the key; the owner of the object constrains who can access it by allowing only particular subjects to have access. The owner states the constraint in terms of the identity of the subject or the owner of the subject.

Definition 7. When a system mechanism controls access to an object and an individual user cannot alter that access, the control is a *mandatory access control (MAC)*, occasionally called a *rule-based access control*.

The system enforces mandatory access controls. Neither the subject nor the owner of the object can determine whether access is granted. Typically, the system mechanism checks information associated with both the subject and the object to determine whether the subject should access the object. Rules describe the conditions under which access is allowed.

Definition 8. An *originator controlled access control (ORCON or ORGCON)* bases access on the creator of an object or the information it contains.

1.2 Motivation and Problem Formulation

Security in a distributed system is a challenging proposition, especially the authorization function of the distributed system's security. The problem of authorization can be divided into two related sub-problems - *representation* and *evaluation*. Representation refers to the specification of authorization

requirements, while evaluation refers to the actual determination of the authorities of the *subjects* given the authorization requirements. The authority of a subject is its rights to access *objects* [1]. The focus of the work described in this thesis is the evaluation of the constraints to check if authorization is granted given the constraints in the context of SPKI/SDSI certificate system in which constraints can be a generic set of constraints represented using logic based formalisms.

Simple Public Key Infrastructure and Simple Distributed Security Infrastructure (SPKI/SDSI) [2] provides a natural framework for authorization in distributed systems. SPKI/SDSI allows distributed security policy definition using public keys and decentralized name space. It is also an elegant practical system that addresses the problem of ensuring that a user is *authorized* to perform an action, not just the problem of identifying the user. In SPKI/SDSI system the certificates are used to ensure confidentiality, integrity and availability. It does incorporate a notion of authentication as well where the linked local namespaces bind keys to names. This notion of authentication is more general than conventional hierarchical PKI naming. The set of certificates that are defined by the security administrator during security cycle of the system being protected partitions the system into the secure and non-secure states described above. The use of certificates during the authorization process ensures that the system is always in 'safe' state. SPKI/SDSI authorization system can be employed in both DAC and MAC scenarios and provide flexible and decentralized authorization infrastructure.

The SPKI/SDSI system as defined in RFC-2693 [2] allows for inclusion of validity period for each certificate. The validity period defines a date pair – *not-before* and *not-after* – within which the certificate is valid. For example, the validity period might specify a validity period as [05/06/2007, 12:00 AM – 10/07/2007, 4:00 PM] which defines the time period in which the validity of the certificate is guaranteed. The definition of validity provides a check on malicious or inadvertent mis-use of the certificate. In addition to time based validity constraints, the SPKI group has also identified that "on-line tests of various kinds" are also validity conditions. These validity conditions further refine the valid date range of a certificate. Three kinds of on-line tests are envisioned: Certificate Revocation List (CRL), Revalidation and One-time.

The RFC-2693 also provides for additional flexibility for performing on-line validity checks on the certificates. Though the RFC provides some pointers to validity condition models to use, it leaves room for inclusion of other models. This provides an opportunity to include additional validation models that caters to new requirements driven by different application and business scenarios. As described by Lee et al. in [3] it may be required to grant authorization decisions based on information of entities that describe their attributes, environmental conditions, and other state information. When these types of systems are modeled in SPKI/SDSI each certificate may be associated with constraints that are based on entity, environment or system specific information. It is possible that the collection of credentials used to satisfy a given authorization policy acts as a partial snapshot of the system within which the policy

is evaluated. The 'snapshot' provides a glimpse of the values of the system at that given moment. This snapshot of value is used to make authorization decision at runtime. As described by Zhang et al. in [4] it is possible that the access control permission on a specific resource is based on dynamic conditions like a user's available usage quota for a particular resource. Current approaches also have limitations when usage status of a shared object is employed for authorization (e.g., the usage context or constraints of resource object). It may also be required that these attributes which are used for authorization decisions are evaluated at runtime so that the last state information is used during evaluation. In addition to this ad-hoc and pervasive collaborations brings new challenges for authorization management. In ad-hoc collaboration subject authentication is not available and authorization decisions are dependent on contextual information [5], such as the location and time of the access request. The approaches known today including those provided SPKI/SDSI system lacks flexibility to support context-based authorizations in collaborative systems. It is possible to envision an access control system facility, similar to that provided by the UCON [4], in which the resource or the associated resource provider decides on usage constraints.

In the era of Internet, in which unknown entities interact with each other to satisfy a request, "trust management" becomes the core issue [6]. In these systems a 'requester' submits a request, possibly supported by a set of 'credentials' issued by other entities, to the resource owner. The authorization decision is taken based on the answer to the following question – "Do the credential provided by the requester prove the request complies with the policies?". The credentials provided may be facts or can be more general non-local policy statement which are much more than facts. The known permission delegation employed in SPKI/SDSI does not allow expression of the fact that the issuer grants permissions to all entities that have a certain property. To simplify authorization in decentralized environments, a system in which access-control decisions are based on authenticated attributes of the subjects is required, and attribute authority is decentralized [6]. In general, the following expressiveness requirements described by Li et al. in [6] are required:

1. The attributes which are evaluated for authorization decision should be decentralized.
2. It shall be possible to delegate the attribute authority to another entity. This is required so that attribute verification can be managed by a single entity.
3. Attribute-based delegation of attribute authority allows use of attribute to distribute the responsibility of attribute authority.
4. Ability to compute conjunction of attributes is required so that an entity can use the conjunction of several attributes to make inferences about another attribute.

It is required that any proposed approach based on SPKI/SDSI not only meet the requirements enumerated above but also confirm to the standard as defined in RFC-2693 [2].

In addition to the requirements and constraints described above an important requirement, common to many applications, is related to the temporal dimension of access permissions. There are many applications and business scenarios in which permissions should hold only for specific time intervals [7]. A further requirement concerns periodic authorizations. In many organizations, authorizations given to users must be tailored to the pattern of their activities within the organization and not just a blank set of authorization that are always applicable. It is essential that users must be given access authorizations to data only for the time periods in which they are expected to need the data to avoid misuse of resources and information. The development of a temporal authorization model entails several issues, including the definition of a formal semantics for the temporal and authorization models, the development of strategies for efficient access control, and the development of tools for authorization administration [7].

An authorization system should not only support positive authorization but should also support authorization which enables explicit denials. Negative authorizations allows exceptions to positive authorizations and for supporting a stricter control in the case of decentralized authorization administration [8]. Additional models can be envisioned when the SPKI certificate system is used not only for access control in the domain of information systems but also in other domains like physical facility access control [9, 10] and agent based transaction systems [11]. These domains require additional constraints that need to be included into the SPKI/SDSI system.

Given all of the above needs the proposal for representation and evaluation should satisfy the following requirements:

- **Rich Temporal Specification:** The validity constraints should enable definition of temporal constraints which is more than just 'from time' and 'to time' specification. Facility to specify temporal constraints like periodicity is required so that the access control policies are specific to the resource access pattern of the user. For example, a constraint like $[3:00PM - 7:00PM\ on\ Mondays]$ define temporal and periodicity constraints together which states that access is provided only within the specified time interval $[3:00PM - 7:00PM]$ and only on the specific day of the week ($Monday$).

- **Attribute Based:** The keys in the SPKI/SDSI can be associated with attributes specific to the entity that the key is associated to. These attributes can be used for defining validity constraints against the certificate. For example, one of the attribute that can be checked is the role that the key is associated to. The authorization authority would use this to selectively grant access permission.

- **Usage Based:** Many critical resources are assigned quota on usage to avoid misuse and manage allocation of usage cost. For these kind of access control systems authorization decisions are taken based on number of times the resource is accessed. After the expiration of the allocated quota the

resource is no longer accessible to the user. An example of this kind of access control can be easily found in the domain of physical facility access control which puts limit on number of user who can simultaneously access the resource. Consider the situation of a conference room which has a maximum occupancy of only 20 people. The usage based access control will not provide access to any new user trying to enter the room once the occupancy of the room has reached its limit.

- **History Based:** A sequence of access to resources forms a history of access for the user. Many a times this history information is used as a basis of subsequent authorization. For example, it may be required that users access resources in a particular predefined order only. This situation is a common-place in the domain of physical facility access control which dictates that the user access a particular room only after valid entry through a sequence of doors.

- **Context Based:** Context-aware computing refers to a computing paradigm in which the behavior of individual components is determined by the circumstances in which they find themselves to an extent that greatly exceeds the typical system/environment interaction pattern common to most modern computing [12]. The context can include state information which are global and information which are local to the user. In the domain of physical facility access the current room that the user is in and the state of the facility as a whole, like emergency situation, forms the context against which authorization decisions are taken. The emerging, pervasive and ubiquitous computing environments need security services that are non-intrusive and easily adaptable to changing user or environmental contexts [13].

- **Spatial:** A spatial access control model discussed here uses the spatial partition as the basis for defining access control constraints. Spatial model of access control is particularly useful in the domain of physical access control when the user physically moves from one spatial partition (room) to another and access resources which are spatially located. Spatial access control model is a very useful concept in the emerging requirement for convergence of physical and logical access control systems. For example, it is possible to envision a physical logical access control system in which access to a resource is provided only when the certificates are available in a particular set of physical locations.

- **Negative Authorization:** A negative authorization specifies the access that must be forbidden, while a positive authorization specifies the access that must be granted. These kinds of authorization explicitly state the constraints that prohibits access to a specific resource. For example, a policy may state that all keys having a specific attribute value are debarred from accessing a resource.

It can be shown that each of the constraints listed above can easily be represented as a regular expression. The alphabet Σ for the regular language represented by the regular expression depends on the type of

constraint modeled. A Weak monadic Second-order Logic of One Successor (WS1S), can be used in place of the formalism of regular expressions. This result is of interest for automata theory because formulas of WS1S seems to be more convenient than regular expressions for formalizing conditions on the behavior of automata [14]. In the following sections it is shown how WS1S can be used to represent constraints like those listed above.

This work specifically attempts to provide solution for the following problems:

- What is the most convenient representation of constraints so that constraints can be defined in a generic way?
- What kind of representation of constraints allows formal analysis and evaluation? Can the known approaches and framework be used for efficient evaluation at runtime?
- How can the constraints that are represented using formal yet user friendly way be associated with known distributed authorization infrastructure like SPKI/SDSI?
- How can the formal models of SPKI/SDSI be modified to accommodate the flexible representations?
- Is the representation and its association with SPKI/SDSI system practically implementable?

1.3 Contributions and Results

This work is primarily focused on formally including more powerful and expressive constraints to SPKI/SDSI system by augmenting an established WPDS framework. The power and expressiveness of the framework known as AWPDS which is an augmentation over WPDS is demonstrated by the implementation of couple of tools.

1.3.1 Flexible constraint definition

A robust constraint definition framework that is based on WS1S is introduced. It is shown how most of the practical constraints that are defined in the literature can be specified using WS1S. The constraints have to be specifically modeled using WS1S based on the access control domain for which the constraint is being modeled. The domain of physical access control is difficult to model due to the physical restrictions imposed by the system and the nature of access control permissions that needs to granted for the physical resources that are protected. The model of secure and non-secure states in the domain of physical access control is different from the traditional access control system in which logical resources like files and directories are protected. WS1S based formulation of access control specification has been provided for the most commonly used constraints. In addition to this a detailed model for a physical

access control has been provided to demonstrate the power of using WS1S for constraint definition. A model for access control specification that is based on order of events is also provided.

1.3.2 Augmentation of WPDS

The Weighted Pushdown System (WPDS) has been augmented so that each rule can be associated with WS1S formula that limits the validity of the rule. The WPDS system described in [15] allows assignment of weights to each rule of PDS. These weights are taken from a bounded idempotent semiring. The ability to associate WS1S formulas to each rule of the WPDS enables richer constraint against each rule. The examples from [15], [16] and [17] have been used to demonstrate the application of the augmentation when temporal constraints are used. In these example the temporal constraints are represented as interval in an abstract domain but the underlying evaluation uses WS1S equivalent of the interval constraints.

1.3.3 The CAP framework

The concept of Comprehensive Authorization Problem (CAP) is formalized, as an extension over the Generalized Authorization Problem (GAP) defined in [15] with additional facility to specify generic constraints represented as regular language. It is demonstrated that various categories of constraints on access control defined in literature can be handled by CAP framework. The algorithm in [15] has been modified so that constraints can be specified using WS1S formula and manipulated using standard automata-theoretic operations. Only the basic algorithm of [15] is modified and the core of the functionality required for answering CAP problems is retained.

1.3.4 A prototype implementation

The algorithm and concepts presented and discussed in this thesis is implemented by building a tool using the MONA [18, 19, 20, 21] and WPDS library of MOPED [22, 23] tools. This tool named SCAT, for **S**PKI/SDSI **C**ertificate **A**nalysis **T**ool, allows analysis and reasoning of SPKI/SDSI certificate set augmented with WS1S specification of temporal constraints, represented as interval over an abstract domain. An XML based input format is used in SCAT which allows definition of the name and authorization certificates along with interval based temporal constraints. The tool uses MOPED WPDS libraries to implement the pushdown system and the facility provided by MONA to model check WS1S formulas. SCAT demonstrates the practical applicability of the concepts described here with the limitation that generic WS1S constraints cannot be provided as input to the current version of the tool. A parser-complier tool has been implemented to demonstrate the feasibility of using WS1S formula in the complex domain of physical access control. This tool named FACT which stands for **F**acility **A**ccess

Constraint **T**ool provides a domain specific and easy to use language which can be syntax checked and converted into a WS1S formula.

Chapter 2

Background Information

2.1 Simple Public Key Infrastructure and Simple Distributed Security Infrastructure(SPKI/SDSI)

2.1.1 Overview

The SPKI/SDSI is a security infrastructure whose principal goal is to facilitate the building of secure, scalable, distributed computing systems [24]. In SPKI/SDSI the *principals are the public keys* and each public key is a certificate authority. Each principal can issue certificates on the same basis as any other principal. There is no hierarchical global infrastructure. SPKI/SDSI communities can be built from the bottom-up, in a distributed manner, and do not require a trusted "root".

There are two types of certificates in SPKI/SDSI – *name certificates* and *authorization certificates*. A name certificate defines a local name in the certificate issuer's local name space, and an authorization certificate grants a specific authorization from the certificate's issuer to the certificate's subject. A single certificate cannot both define a name and grant an authorization. Each certificate is either strictly a name certificate or an authorization certificate.

In the following $\mathcal{K}$ denotes the set of public keys and specific keys are denoted by K, K_A, K_B, K', ..., etc. An *identifier* is a word over some alphabet Σ. The set of identifiers is denoted by $\mathcal{A}$. Identifiers are written in typewriter font, e.g. `A` and `Bob`.

A *term* is a key follow by 0 or more identifiers. Terms are either keys, local names, or extended names.

A *local name* is of the form K `A`, where $K \in \mathcal{K}$ and $\texttt{A} \in \mathcal{A}$ is an identifier. For example, K `Bob` is a local name. Local names are important in SPKI/SDSI because they create a decentralized name space. The set of all local names is denoted by $\mathcal{N}_L$, and the local name space of K (local names of the form

K `A`) is denoted by $\mathcal{N}_L(K)$.

An *extended name* is of the form $K\ \sigma$, where $K \in \mathcal{K}$ and σ is a sequence of identifiers of length greater than one. For example, K UW `CS faculty` is an extended name. Let $\mathcal{N}_E$ be the set of extended names and $\mathcal{N}_E(K)$ denote the set of extended names beginning with key K. The set of names in $\mathcal{N}$ is $\mathcal{N}_L(K) \cup \mathcal{N}_E(K)$. The set of terms $\mathcal{T}$ is thus $\mathcal{K} \cup \mathcal{N}$.

2.1.2 Certificates

SPKI/SDSI has two types of certificates, or "certs". The first type of certificate, called *name certs*, provides definitions of local names. Authorization are specified using *authorization certs* or *auth certs*, for short.

2.1.2.1 Naming

SPKI/SDSI *name certificates* bind local names to public keys. Assuming each user has a single public-private key pair, the name-to-key binding is a multi-valued function: each name is bound to zero, one or more keys. A single name certificate can define a name in the issuer's local name space to be a public key, another name in his/her local name space, or a name in another principal's local name space. A name certificate that defines the local name "K `A`", where K is the issuer's key, to be the subject, "`S`", can be denoted as "K `A` $\longrightarrow$ `S`".

As each principal can issue name certificates, each principal has its own local name space, consisting of the names it defines. SPKI/SDSI, thus, has a local name space architecture, which helps to make the infrastructure scalable: a user does not have to ensure that the names they define are unique in a global name space.

A name certificate consists of four fields: the issuer's key, an identifier, the certificate's subject, and a validity specification. An identifier is a single word over some standard alphabet, such as `Alice`, `Bob`, `Friends`, `A`, `B`. Following are descriptions of these certificate fields:

- **Issuer:** The public key that signs the certificate.
- **Identifier:** The identifier determines the local name that is being defined. The name being defined consists of the issuer's key and this identifier. A name certificate, thus, defines a name that consists of a single key followed by a single word. A name consisting of a single key followed by exactly one identifier is referred to as a *local name*. Because a name certificate can only define a local name, each principal can only define names within its own name space.
- **Subject:** The new meaning of the local name being defined. A subject can be a public key or a name consisting of a single public key followed by one or more identifiers. The public key in the subject does not have to be the issuer's key.

- **Validity Specification:** The validity specification is a time period during which a certificate is valid, assuming the signature verifies. Beyond this period, the certificate has expired, and should be renewed. The validity specification takes the form (t_1, t_2), specifying that the certificate is valid from time t_1 to time t_2, inclusive.

2.1.2.2 Group

A SPKI/SDSI *group* is typically a set of principals. Each group has a name and a set of members. The name is local to some principal, who is the "owner" of the group, and the group owner is the only one who can change the definition of the group. A group definition may explicitly reference the members of the group, or reference other groups (which may even belong to someone else.) To define a group, a group owner issues, to each group member, a name certificate defining the local name of the group in the owner's name space to be that member's key or name. A group owner can also add any principal's group to his group by issuing name certificates binding the name of his group to the name of the group being added. Figure 2.1 gives an example of a SPKI/SDSI group.

$$K_A \texttt{ friends} \longrightarrow K_B \tag{2.1}$$
$$K_A \texttt{ friends} \longrightarrow K_C \tag{2.2}$$
$$K_A \texttt{ friends} \longrightarrow K_D \tag{2.3}$$
$$K_A \texttt{ friends} \longrightarrow K_E \texttt{ E-Friend} \tag{2.4}$$
$$K_A \texttt{ friends} \longrightarrow K_B \texttt{ B-Sister friends} \tag{2.5}$$

Figure 2.1: An example of a SPKI/SDSI group: K_A friends

The ability to create groups makes security policies easier and more intuitive to define as they can be explicitly specified in terms of groups. As the names of the groups are at the discretion of the owners, groups can have meaningful, intuitive names. This makes the auditing of group definitions and ACLs simpler. One disadvantage of groups is that revoking the membership of a group member is non-trivial. If an explicit list of principals is maintained on the ACLs, the untrustworthy member's privileges can be easily revoked by removing his key from the ACLs.

2.1.2.3 Authorization

The SPKI/SDSI *authorization certificate* grants a specific authorization from the certificate's issuer to the certificate's subject. A SPKI/SDSI authorization certificate consists of five fields: the issuer's key, the certificate's subject, a delegation bit, a tag, and a validity specification. Following are descriptions of these fields:

- **Issuer:** The key that signs the certificate. This issuer is the principal granting the specific

authorization.

- **Subject:** The key or group that is receiving the grant of authorization. A subject can be a public key or a name consisting of a single public key followed by one or more identifiers. The public key in the subject does not have to be the issuer's key.
- **Tag:** The tag specifies the specific authorization or authorizations being granted from the issuer to the subject.
- **Delegation Bit:** If this bit is true, the subject of this certificate is able to grant to other principals any subset of the authorization that it is receiving from the issuer of this certificate. If this bit is false, the issuer is not delegating to the subject the authority it is granting to it.
- **Validity Specification:** This is the same as that for a name cert.

An authorization certificate in which the issuer, "K", grants the authorization specified in the tag, "T", to the subject, "S", with the delegation bit, "p", and a validity specification, "V", is represented by the 5-tuple, "(K, S, T, p, V)". The value of "p" is either true (□) or false (■). This can be denoted as "K□ $\xrightarrow{\texttt{T}}$ S □" if p=true otherwise "K □ $\xrightarrow{\texttt{T}}$ S ■" .

SPKI/SDSI provides a very simple, flexible, authorization model. Authorizations can be specified as precisely or as generally as desired using flexible, user-defined tags. *Intuitively, a tag represents a set of requests.*

The ability to specify authorizations in tags in certificates is a powerful notion. Conventional certificates principally bind names to keys. However, a user's name is only one attribute of the user, and is rarely of security interest. An administration needs to know whether a user has been granted specific authorization to access the protected resource, and, if so, who granted that authorization.

A way of considering an authorization certificate is that it *transfers* or *propagates* a specific authorization from the issuer to the subject. If the delegation bit is set in the certificate, the subject is allowed to continue propagating this authorization, or some subset of it, to other principals, by issuing authorization certificates to them. If the subject, in turn, sets the delegation bit on its certificates to true, the principals to whom it is propagating the authorization will also be able to issue authorization certificates granting new principals the authority, and so on.

2.1.3 The Authorization Problem in SPKI/SDSI

Let $\mathcal{C}$ be a set of certs and K' and R are the public keys of a client and resource owner, respectively, with authorization specification on the resource R is T'. In the domain of authorization a problem that needs to be solved is the question: "Given a request $r = (K', R, T')$, is K' allowed to exercise authorization T' on R?" A *certificate-chain-discovery* algorithm provides not only a *Yes/No* answer to the authorization

question but also a proof if one exists. In the case of a *Yes* answer, certificate-chain-discovery algorithm identifies a chain of certificates to prove the result. Formally, certificate-chain discovery attempts to find, after removing useless certificates, a certificate chain $c_k \circ \cdots \circ c_1$ such that

$$(c_k \circ \cdots \circ c_1)(R\Box) \in \{K'\Box, K'\blacksquare\}.$$

Intuitively, $(c_k \circ \cdots \circ c_1)$ represents a path from R, the resource, to either $K'\Box$ or $K'\blacksquare$, representing "permission for K' to access" with and without delegation, respectively; the elimination of useless certs ensures that the chain represents the authorization specification T'.

Clarke et al. [25] presented an algorithm for certificate-chain discovery in SPKI/SDSI with $O(n_K^2|\mathcal{C}|)$ time complexity, where n_K is the number of keys and $|\mathcal{C}|$ is the sum of the lengths of the right-hand sides of all rules in $\mathcal{C}$. However, this algorithm only solved a restricted version of certificate-chain discovery: a solution could only consist of a single certificate chain. Schwoon et al. [15] presented algorithms for full certificate-chain discovery, based on solving reachability problems in weighted pushdown systems. Their formalization allows a proof of authorization to consist of a set of certificate chains. The work presented here uses the WPDS-based algorithm for distributed certificate-chain discovery introduced by Schwoon et al. in [16] as it allows exploration of all valid certification chains that prove the availability of authorization to access a specific resource.

2.1.4 Authorization Flow

The SPKI/SDSI infrastructure is primarily concerned with authorizing principals to perform particular operations on protected resources. To provide access control on a resource, the guardian sets up an access control list (ACL) to protect it. Let K_c be a typical user requesting to perform a particular operation on the resource K_r. Examples of requests are a request to read a particular file, a request to read a file in a particular directory, a request to login to a particular account, and a request to turn on a particular electrical appliance or request to enter a particular room; in these examples, the 'protected resources' are the file, directory, account, appliance and room, respectively. For the resource owner (provider) to honor K_c's request, her request must be accompanied with a "proof of authenticity", that authenticates the request, and a "proof of authorization" that shows that she is authorized to perform the request. The "proof of authenticity" is typically a signed tag, and the "proof of authorization" is typically a sequence/chain of certificates. The principal that signed the tag must be the same principal that the chain of certificates authorizes.

An authorization chain/flow consists of a chain of valid certificates that authorizes K_c to perform the specific operation she is requesting to perform in the tag she signed. The proof of authorization, i.e. certificate chain, in K_c's request provides an authorization chain from the ACL's issuer to her public

key, the key that signed the request. When the K_r verifies the certificate chain, it is verifying that there is a chain of authorization originating from it, through the ACL, through zero or more certificates, to K_c's key. If K_r successfully verifies this authorization chain, K_c is authorized to perform the requested operation on the protected resource. The SPKI/SDSI certificate chains which demonstrate flows of authorization can consist of just authorization certificates or both name certificates and authorization certificates. When K_c is being issued her certificates, she should be given all of the certificates necessary to establish the necessary chain of authorization.

To understand and analyze a set of certificates and the chain that they form for a specific client, resource-owner pair a formal model is required. One of the formal model of SPKI/SDSI system is Weighted Pushdown System (WPDS) which is described in the next section. This model is also used for solving the authorization and certificate chain discovery problems.

2.2 Weighted Pushdown Systems (WPDS)

The formalism of Weighted Pushdown System (WPDS) is described briefly and its connection to SPKI/SDSI system is explained. The algorithm for certificate-chain discovery given in [26] is explained. The concept of WPDS is an extension over Pushdown Systems (PDS).

2.2.1 Pushdown Systems

[16] A pushdown system is a transition system equipped with a finite set of control locations and a stack. The stack contains a word over some finite stack alphabet and its length is unbounded. Hence, a pushdown system may have infinitely many reachable states.

Definition 9. A **Pushdown System** is a triple $\mathcal{P} = (P, \Gamma, \Delta)$, where P and Γ are finite sets called the **control locations** and the **stack alphabet**, respectively. The elements of $Conf(\mathcal{P}) := P \times \Gamma^*$ are called the **configurations** of $\mathcal{P}$. Δ contains a finite number of rules of the form $\langle p, \gamma \rangle \hookrightarrow_{\mathcal{P}} \langle p', w \rangle$ which define a transition relation between configurations of P as follows:
if $r = \langle p, \gamma \rangle \hookrightarrow_{\mathcal{P}} \langle p', w \rangle$, then $\langle p, \gamma w' \rangle \Longrightarrow_{\mathcal{P}} \langle p', ww' \rangle$ for all $w' \in \Gamma^*$.
$c \Longrightarrow_{\mathcal{P}} c'$ is written to express that there exists some rule r such that $c \overset{\langle r \rangle}{\Rightarrow} c'$; omit the subscript $\mathcal{P}$ if $\mathcal{P}$ is understood. The reflexive transitive closure of $\Rightarrow$ is denoted by $\Rightarrow^*$. Given a set of configurations C, define $pre^*(C) \overset{def}{=} \{c' \mid \exists c \in C : c' \Rightarrow^* c\}$ and $post^*(C) \overset{def}{=} \{c' \mid \exists c \in C : c \Rightarrow^* c'\}$. Given a pushdown system $\mathcal{P} = (P, \Gamma, \Delta)$, a *regular* set of configurations of $\mathcal{P}$ can be represented with a finite-state automaton, called a *configuration automaton* of $\mathcal{P}$, whose input alphabet is $\mathcal{P}$'s stack alphabet.

Formally, a configuration automaton of $\mathcal{P}$ is an automaton $\mathcal{A} = (\Gamma, Q, \delta, P, F)$, where Q is a finite set of states and the set of locations P of $\mathcal{P}$ is a subset of Q, $\delta \subseteq Q \times \Gamma \times Q$, is the set of *transitions*; P is

the set of initial states; and $F \subseteq Q$ is the set of final states. The configuration automaton's reachability relation, denoted by $\xrightarrow{w} \subseteq Q \times \Gamma^* \times Q$, is defined as the smallest relation satisfying:

- $q \xrightarrow{\epsilon} q$ for every $q \in Q$
- $(q, \gamma, q') \in \delta$, then $q \xrightarrow{\gamma} q'$, and
- if $q \xrightarrow{w} q''$ and $q'' \xrightarrow{\gamma} q'$, then $q \xrightarrow{w\gamma} q'$

A configuration automaton *accepts* or *recognizes* a configuration $\langle p, w \rangle$ of $\mathcal{P}$ if $p \xrightarrow{w} q$ for some $p \in P$ and $q \in F$. The set of configurations recognized by automaton $\mathcal{A}$ is denoted by $Conf(\mathcal{A})$.

2.2.2 Weighted Pushdown Systems (WPDS)

Weighted pushdown systems (WPDS) were introduced in [26, 27, 15]. A pushdown system defines an infinite-state transition system whose states involve a stack of unbounded length. In a weighted pushdown system, the rules are given values from some domain of weights. The weight domains of interest are the bounded idempotent semirings defined in Definition 10.

Definition 10. A **bounded idempotent semiring** is a quintuple $(D, \oplus, \otimes, 0, 1)$, where D is a set, 0 and 1 are elements of D, and $\oplus$(the combine operation) and $\otimes$(the extend operation) are binary operators on D such that

1. $(D, \oplus)$ is a commutative monoid whose neutral element is 0, and where $\oplus$ is idempotent.
2. $(D, \otimes)$ is a monoid with the neutral element 1.
3. $\otimes$ distributes over $\oplus$, i.e., for all $a, b, c \in D$ we have $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ and $(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$.
4. 0 is an annihilator with respect to $\otimes$, i.e., for all $a \in D$, $a \otimes 0 = 0 = 0 \otimes a$.
5. In the partial order $\sqsubseteq$ defined by: $a, b \in D$, $a \sqsubseteq b$ iff $a \oplus b = a$, there are no infinite descending chains.

Definition 11. A **weighted pushdown system** is a triple $\mathcal{W} = (\mathcal{P}, \mathcal{S}, f)$ such that $\mathcal{P} = (P, \Delta, \Gamma)$ is a pushdown system, $\mathcal{S} = (D, \oplus, \otimes, 0, 1)$ is a bounded idempotent semiring, and $f : \Delta \to D$ is a function that assigns a value from D to each rule of $\mathcal{P}$. Let $\sigma \in \Delta^*$ be a sequence of of rules. Using f, we can associate a value to σ, i.e., if $\sigma = [r_1, \ldots, r_k]$, then we define $v(\sigma) \stackrel{def}{=} f(r_1) \otimes, \ldots, \otimes f(r_k)$. For any two configurations c and c' of $\mathcal{P}$, let $path(c, c')$ denote the set of rule sequence $[r_1, \ldots, r_k]$ that transforms c into c', i.e., $c \xRightarrow{\langle r_1 \rangle} \ldots \xRightarrow{\langle r_k \rangle} c'$

Definition 12. Let $\mathcal{W} = (\mathcal{P}, \mathcal{S}, f)$, where $\mathcal{P} = (P, \Delta, \Gamma)$ and $\mathcal{S} = (D, \oplus, \otimes, 0, 1)$, and let C bet a set of configurations. A forwards (respectively backwards) $(\mathcal{W}, C)$-dag is an edge-labeled directed acyclic graph (V, E) where $V \subseteq \text{Conf}(\mathcal{P}) \times D$ and $E \subseteq V \times \Delta \times V$ such that

- if a vertex (c, d) has no incoming edges, then $c \in C$ and $d = 1$;
- if $((c_1, d_1), r_1, (c, d)), \ldots, ((c_k, d_k, r_k, (c, d))$, $k \geq 1$ are the incoming edges of (c, d), then
  - $d = \bigoplus_{i=1}^{k}(d_i \otimes f(r_i))$ and $c_i \stackrel{\langle r_1 \rangle}{\Longrightarrow}_{\mathcal{P}} c$ for all $1 \leq i \leq k$ (in a forwards $(\mathcal{W}, C)$-dag);
  - $d = \bigoplus_{i=1}^{k}(f(r_i) \otimes d_i)$ and $c \stackrel{\langle r_1 \rangle}{\Longrightarrow}_{\mathcal{P}} c_i$ for all $1 \leq i \leq k$ (in a backwards $(\mathcal{W}, C)$-dag).

A forwards/backwards $(\mathcal{W}, C)$-dag is called a witness dag $\mathcal{D}$ for (c, d) if $\mathcal{D}$ is finite and (c, d) is the only vertex with no outgoing edges in $\mathcal{D}$.

The extender operation $\otimes$ is used to calculate the value of a path. The value of a set of paths is computed using the combiner operation $\oplus$. The existence of a witness dag for (c, d) can be considered a proof that there exists a set of paths from C to c (or vice versa) whose combined value is d.

Definition 13. Let $\mathcal{W} = (\mathcal{P}, \mathcal{S}, f)$, where $\mathcal{P} = (P, \Delta, \Gamma)$ and let $C \subseteq P \times \Gamma^*$ be a regular set of configurations. The **Generalized Pushdown Predecessor (GPP) problem** is to find for each $c \in pre^*(C)$:

- $\delta(c) \stackrel{def}{=} \oplus\{v(\sigma) | \sigma \in path(c, c'), c' \in C\}$
- a backwards witness dag for $(c, \delta(c))$.

In [15, 26], the solutions for Generalized Pushdown Predecessor (GPP) are computed in the form of annotated finite automata in which the transitions are annotated with the weights.

2.2.3 The Connection Between SPKI/SDSI and Weighted Pushdown Systems

The formalism of WPDS and the specification of SPKI/SDSI can be related. This relation allows use of the formalism for reasoning about SPKI/SDSI based systems.

Given a finite set of certificates $\mathcal{C}$ such that $\mathcal{K}_\mathcal{C}$ and $\mathcal{I}_\mathcal{C}$ are the keys and identifiers that appear in $\mathcal{C}$, respectively and also given the set $\mathcal{T}$ from which the auth specs in $\mathcal{C}$ are drawn, the correspondence between WPDS and SPKI is given as follows:

- $\mathcal{S}_\mathcal{C} = (\mathcal{T}, \cup, \cap, \bot, \top)$ where $\cup$ and $\cap$ are the union and intersection of auth specs as discussed in [2, 28], forms a semiring with domain $\mathcal{T}$.
- associate with $\mathcal{C}$ the weighted pushdown system $\mathcal{W}_\mathcal{C} = (\mathcal{P}_\mathcal{C}, \mathcal{S}_\mathcal{C}, f)$, where $\mathcal{P}_\mathcal{C} = (\mathcal{K}_\mathcal{C}, \mathcal{I}_\mathcal{C} \cup \{\square, \blacksquare\}, \Delta_\mathcal{C})$.

- the keys of $\mathcal{C}$ are the control locations and the identifiers form the stack alphabet;
- the rule set $\Delta_{\mathcal{C}}$ is defined as the set of labeled rewrite rules and f maps every rule to its corresponding auth spec.

Given the above correspondence between WPDS and SPKI/SDSI a configuration $\langle K, \sigma \rangle$ of $\mathcal{P}_{\mathcal{C}}$ can reach another configuration $\langle K', \sigma' \rangle$ if and only if $\mathcal{C}$ contains a chain of certificates $c_1, \ldots, c_k$ such that $(c_k \circ \ldots \circ c_1)(K\sigma) = (K'\sigma')$. The label of the certificate chain is precisely $v(c_1, \ldots, c_k)$. Thus, solving the GPP problem amounts to finding a set of certificate chains to prove that a certain principal K' is allowed to access a resource of principal K. The solution of the problem identifies a set of certificate chains such that the union of their labels is maximal.

The generalized authorization problem (GPP) in SPKI/SDSI is cast as follows: "Given a set of certificates $\mathcal{C}$, a resource owner K_r, an authorization specification T, and a client K_c, are there certificate chain in $\mathcal{C}$ proving that K_r grants authorization T to K_c?"

The GPP question is recast into an equivalent question in the WPDS setting as there exists a connection between SPKI/SDSI system and WPDS as demonstrated above. The equivalent GPP problem in the WPDS setting is: "Given $C = \{\langle \mathcal{K}_r, \blacksquare \rangle \langle \mathcal{K}_c, \square \rangle\}$ and $c = \langle \mathcal{K}_r, \square \rangle$, compute $t := \delta(c)$ and a backwards witness dag for $(c, \delta(c))$". Authorization for K_c is granted if and only if $t \supseteq T$.

2.3 Weak monadic Second-order Logic of One Successor(WS1S)

The Weak monadic Second-order Logic of One Successor (WS1S) is a variation of first order logic with the difference that it's interpretation is tied to arithmetic, somewhat weakened to keep the formalism decidable. In WS1S, first-order variables denote natural numbers, which can be compared and subjected to addition with constants. WS1S also allows second-order variables while remaining decidable; each such variable is interpreted as a finite set of numbers. WS1S is a logic interpreted over natural numbers, $\mathbb{N}_0 = \{0, 1, \ldots\}$. The quantification is over individual elements of $\mathbb{N}_0$ and subsets of $\mathbb{N}_0$ which follows natural ordering of $\mathbb{N}_0$ (unique and one successor).

WS1S logic can help in practice to identify and to use regularity. In this logic it is possible to directly mention positions and subsets of positions in the string. This feature distinguishes the logic from regular expressions or automata. Together with quantification and Boolean connectives, an extraordinary succinct formalism arises.

2.3.1 Syntax

The syntax of WS1S consists of Terms, Atomic Formulas and Formulas. A *term* is built up from constant 0 and individual variables $x, y, \ldots$ by application of successor function *succ*. Examples of terms are – 0, $succ(x)$, $succ(succ(succ(67)))$, $succ(succ(y))$.

An *atomic formula* is of the form $t = t'$ or $t \in X$ where t and t' are terms and X is a set variable. A *formula* is built up from atomic formulas using the Boolean connectives $\neg$(not), $\vee$(or) with the existential quantifier ($\exists$). Existential quantifier ($\exists$) can be applied to both individual variables and set variables. Examples of formulas include: $\neg\varphi$, $\varphi \vee \psi$, $(\exists x)\varphi$ and $\exists X(\varphi)$. Remaining Boolean connectives are defined using $\neg$(not) and $\vee$(or). For example – $\varphi \wedge \psi$ is defined as $\neg(\neg\varphi \vee \neg\psi)$; $\varphi \Rightarrow \psi$ is defined as $(\neg\varphi \vee \psi)$; $\varphi \leftrightarrow \psi$ is defined as $(\neg\varphi \vee \psi) \wedge (\neg\psi \vee \varphi)$. Universal quantifier $\forall$ is defined using $\exists$. $(\forall x)\varphi$ is define as $\neg((\exists x)\neg\varphi)$ and $(\forall X)\varphi$ is defined as $\neg((\exists X)\neg\varphi)$. A few example of formulas include:

- $x \notin X$ is defined as $\neg x \in X$
- $X = Y$ is defined as $(\forall x)[(x \in X \Rightarrow x \in Y) \wedge (x \in Y \Rightarrow x \in X)]$
- $Sub(X, Y)$ is defined as $(\forall x)(x \in X \Rightarrow x \in Y)$
- $Zero(x)$ is defined as $(\exists x)[(x \in X) \wedge \neg(\exists y)(y < x)]$
- $Sing(X)$ is defined as $(\exists Y)[Sub(Y, X) \wedge (Y \neq X) \wedge \neg(\exists Z)(Sub(Z, Y) \wedge (Z \neq Y))]$
- $Lt(x, y)$ is defined as $(\forall Z)[succ(x) \in Z \wedge (\forall Z)(z \in Z) \Rightarrow (succ(z) \in Z)] \Rightarrow (y \in Z)$

2.3.2 Semantics

The semantics of WS1S is such that formulas are interpreted over $\mathbb{N}_0$. Individual variables $x, y, \ldots$ are interpreted as natural numbers i.e. elements of $\mathbb{N}_0$. Function *Successor* corresponds to adding one. $t = t'$ is true provided t and t' denote the same natural number. Set variables like $X, Y, \ldots$ are interpreted as subsets of $\mathbb{N}_0$. $t \in X$ is true iff the number denoted by t belongs to the set denoted by X. A variable is said to occur free in a formula if it is not within the scope of a quantifier. Variables which do not occur free are said to be bound. For example: $(\exists x)[(x \in X) \wedge \neg(\exists y)(y < x)]$, x and y are bound variables and X is a free variable. $\varphi(x_1, x_2, \ldots, x_k, X_1, X_2, \ldots, X_l)$ indicates all the variables which occur free come from $\{x_1, x_2, \ldots, x_k, X_1, X_2, \ldots, X_l\}$. To assign a truth value to the formula $\varphi(x_1, x_2, \ldots, x_k, X_1, X_2, \ldots, X_l)$, map each individual variable x_i to a natural number $m_i \in \mathbb{N}_0$ and each set variable X_j to a subset $M_j \subseteq \mathbb{N}_0$. $M \vdash \varphi(X)$ denote that φ is true under the interpretation $\{x_j \rightarrow m_j\}_{i \in \{1,2,\ldots,k\}}$ and $\{X_i \rightarrow M_i\}_{i \in \{1,2,\ldots,l\}}$. For example: $(M, N) \models Sub(X, Y)$ iff $M \subseteq N$; $M \models Zero(X)$ iff $0 \in M$; $(m, n) \models Lt(x, y)$ iff $m < n$; $M \models Sing(X)$ iff M is a singleton $\{m\}$.

A sentence is a formula in which no variables occur free. A sentence is either true or false. Assigning values is not needed for a sentence. An example of a sentence is: $\forall X[0 \in X \wedge (\forall x)(x \in X \Rightarrow succ(x) \in X)] \Rightarrow (\forall x)(x \in X)$.

An WS1S formula $\varphi(x_1, x_2, \ldots, x_k, X_1, X_2, \ldots, X_l)$ is said to be satisfiable if we can choose $M_1 = (m_1, m_2, \ldots, m_k, M_1, M_2, \ldots, M_l)$ such that $M_1 \models \varphi(X_1)$, where $X_1 = (x_1, x_2, \ldots, x_k, X_1, X_2, \ldots, X_l)$.

Büchi in [14] showed that every word in L_φ ,which is the language corresponding to the WS1S formula φ, has an interpretation for the free variables in φ under which φ evaluates to true. Every interpretation which makes φ true is represented by some word in L_φ. φ is satisfiable iff there is some interpretation which makes it true iff L_φ is nonempty. The language L_φ is defined over the alphabet $\{0,1\}^m$ where m is the number of free variables in φ. Given a WS1S formula φ an automata that accepts L_φ can be constructed inductively and the set of satisfying interpretations of a subformula is represented by a finite-state automaton. WS1S thus acts as a notation for regular languages, just as regular expressions do.

Chapter 3

Related Work

The work presented in this thesis is based on the prior work accomplished in the fields of model checking of SPKI/SDSI certificate system, certificate chain discovery in SPKI/SDSI and the distributed version of the same, Pushdown System and its extensions and other similar work related to SPKI/SDSI certificate system. In addition to this, previous results obtained in research pertaining to Decentralized Access Control is also cited and described here as they provide the foundation for the work in the field of using WS1S for constraint specification in access control domain.

3.1 Certificate Chain Discovery in SPKI/SDSI

A certificate-chain-discovery algorithm for SPKI/SDSI was first proposed by Clarke et al. in [25]. The certificate chain discovery algorithm presented takes an initial set $\mathcal{C}$ of certificates, an authorization that is desired and and public key K for which it is desired to prove that authorization exists. The proof consists of a chain of certificates that proves that K is authorized to access the requested resource given the set of certificates in $\mathcal{C}$. The certificate chain discovery is the core of the work described in this thesis. The process of certificate chain discovery has been extended so that flexible constraints can be defined.

An improved certificate chain discovery based on the theory of pushdown systems (PDS) was presented by Jha and Reps [29]. The connection between SPKI/SDSI system and PDS has been demonstrated and certificate chain analysis problem has been solved using PDS as formalism. Jha and Reps also demonstrate how several useful questions about authorization can be solved by annotating the PDS and configuration automaton with labels from a lattice. The use of lattice restricts the authorization problems to those set in which constraints can be specified and reasoned using lattice. Both of the algorithms in [25] and [29] are centralized and assume that the proof of authorization consists of a single certificate chain. In general, a proof of authorization in SPKI/SDSI requires a set of certificate chains, each of which proves some part of the required authorization.

The semantics of SPKI/SDSI has also been studied in [28, 30, 31]. In the proof-carrying-authorization (PCA) framework of Appel and Felten [32], a client uses the theorem prover Twelf [33] to construct a proof of authorization, which the client presents to the server. However, in this too it is assumed that all logical facts used by theorem prover reside at a single server/location. Li et al. in [34] present a distributed credential chain-discovery algorithm for the trust management system RT0. Their algorithm allows credentials to be distributed, but the proof of authorization is constructed at one site. The credential-chain-discovery algorithms of Li et al. fetches credentials from other sites as needed. Formalising the semantics of SPKI/SDSI system is of importance for evaluation of the certificate set and answering authorization questions pertaining to specific access requests.

In all of the above cited work the constraint specification mechanism only allows for specific instances of constraint specification. The most common constraint that is specified against the rules or policies in the access control system is temporal constraints.

3.2 Context Dependent Access Control

The results on context-dependent access control is described here to understand the formalism of context-dependent access control as discussed in the literature and the approaches that have been taken to solve this problem. This understanding is relevant to the work described here to contrast with the approach of using WS1S formula for specifying constraints that are based on the context of the system and the user.

Ruben et al. in [35] describe a context-dependent access control for web services with role-based approach. In the method described the permissions that are assigned to the role is dependent upon the identification mechanism used by the user (which defines the context of access), which implies that the set of permissions that are assigned to the role to which the user is associated depends upon the identification mechanism, unlike the challenge-response or the password based approach, used during the authentication phase. If a user is assigned to several role(s), it is up to the user to decide which role(s) he is authorized to activate. The activation depends on the way the user identifies himself. This can be seen as a user indirectly selecting a role to activate. This in-turn means that based on tje identification aspects, the user role to be activated is selected by the access control system. The definition of context in the approach described in this thesis is the state of the system or the user. The current state of the system and the user defines the constraints within which access permissions are granted to the user.

In Junzhe et al. [36] describe the security requirements in healthcare system which require very dynamic and flexible policy enforcement. In the proposed security infrastructure, the policy enforcement is highly dynamic and independent from any particular application. A context constraint defined as a regular expression is allowed in the infrastructure which allows specification of complex context

related constraints to describe all kinds of security requirements. The access control scheme also consists of authorization policy and data access specification which together with dynamic context evaluation algorithm decides whether an access is authorized or not based upon the context type and its implementation. Georgiadis and Mavridis [37] and Wang [38] both present a team-based access control model that is aware of contextual information associated with activities in application. Kumar et al. summarizes previous work and formally proposes context-sensitive RBAC model. The results presented in [36] is very relevant to the work described in this thesis as both the approaches use regular languages to describe the access constraints. The approach described in this work is different in the use of WS1S for modeling and representation of constraints and use of automata theoretic operators to extend or constrict the constraints. The framework proposed here also includes the constraints with SPKI/SDSI system so that the power of SPKI/SDSI authorization infrastructure is used.

The access control problem for mobile agents is described in [39]. The access control function (ACF) takes five parameters: the credentials, operation, tuple, pattern, and the owner's profile and determines whether or not to allow the requested access. Formally, this function can be represented as: $ACF : T \times C \times O \times P \times \Pi \rightarrow \{0, 1\}$, where T is the universe of tuples, C is the universe of credentials, O is the finite set of operations, P is the universe of patterns, and Π is the universe of profile. The access control function maps the values of the parameters to a boolean indicating the access decision. The context-sensitivity is in P, the universe of patterns. The pattern P used here for context-dependence can be related to the pattern expressed as regular language the approach presented in this thesis. The function ACF which maps the tuples T to access rights $\{0, 1\}$ is a specific instance of mapping from constraints to certificates described in the results presented in this thesis.

A framework for representing the behavior of access control policies in dynamic environment is presented by Daniel et al. in [40]. This paper presents formal analysis for access control policies in their dynamic environment. A mathematical model of policies, their environment, and the interaction between them is proposed. The policies of interest are captured as Datalog programs. Similar to the approach used in the framework proposed in this thesis, the principal source of environment information is defined as *event* which could come from user end-users, run-time systems and policy framework itself. Transitions in the policy's environment are triggered by various conditions. Some arise from the passage of time and other from user or program actions. A policy in this framework interacts with its dynamic environment by consulting facts in the environment and potentially constraining certain actions in the environment. The latter captures influence of policy decisions on an application that uses it. A policy and an environment model for the policy's dynamic environment combine to form a state machine over access tables.

Meenakshi and Arul Ganesh et al. in [9], [10] describe a system for context based access control in which the policies are encoded as automata and the transitions are labeled with events which defines

the context of the system. The transition system that is described has event composition method that uses Prolog for evaluation of context. The authors describe two types of contexts that are handled by their framework – user context and system context. Both of these contexts for provided as events for the automata based access control system to react. The WS1S based specification of access control specification is used as the basis for definition of constraints.

3.3 Temporality in Access Control

A new model for representing temporal access control (TIAC) policies is introduced in [41]. In this model, temporal authorizations are represented by time attributes associated with both subjects and objects, and a "time interval access graph". The time interval access graph is used to define constraints on the temporal relations between subjects and objects. Interval algebra is used to define and analyze the time interval access graph. The TIAC model provides formal semantics to express temporal authorization policies, in which temporal attributes of subjects and objects are used to determine authorized accesses. This is the first know use of interval algebra to express a temporal access control policy. The interval algebra provides necessary expressive power to logically describe a temporal access control policy, and a precise and efficient way to computationally reason about the temporal relation between subjects and objects and associated access constraints. In the model proposed, time is assumed to be a set of discrete points, $\mathbf{T}$, which is isomorphic to the natural numbers and is linearly ordered with respect to the '$<$' relation. Points in $\mathbf{T}$ are used in representing time intervals. Time intervals are represented using half-open intervals denoted as $\tau = [t-, t+)$ where $t- < t+$. Half-open intervals are used so that there are not semantic ambiguities about the point where two time intervals meet. Time intervals are associated with subjects and objects, and temporal access control policies are reasoned about using interval algebra. Subjects and objects each have an associated time interval attribute, which is used for making access control decisions. The TIAC model introduces the time interval access graph which is consistent instantiation of the three-vertex interval algebra network that defines access constraints on the temporal relations between subjects and objects, and a reference time interval (τ_{ref}). The access requests by a temporal subject to an object is decided based on the access mode in the time interval access graph. The work is relevant as it formally includes time intervals during the access control decision process unlike other models in which time intervals are not so closely bound with the access control process. The approach described in this thesis uses a temporal model which is similar to the time intervals described in [41]. The reasoning of time has been done using automata theoretic approach and the definition of the temporal interval is done using WS1S formula.

In [7] the authors consider the problem of granting/denying authorizations with respect to time. They develop ways to model access control policies that vary with time and to specify policies that are

applicable periodically. Time is modeled symbolically as follows:

- "Calender" is defined to be a countable set of consecutive intervals. For example, assuming hours as the basic calender (i.e., one unit), days can be defined as $days = generate(1; hours; (24))$. Similarly, months, years etc. are defined as various calenders.
- "Periodic expressions" defining infinite sets of "periodic intervals" are defined. These are the usual time intervals, having a starting point, an ending point and a period.
- It is argued that periodic expressions and intervals are not very convenient from the point of view of implementation and therefore define "gap order constraints" and "periodicity constraints" as a solution. These are basically relations involving $<$, $=$, integer constants and variables. For ease of representation and manipulation, their normal form is represented by a "temporal constraint", which is a propositional logic formula over periodicity constraints and gap order constraints.

The authors show that any symbolic periodic expression can be translated into an equivalent set of simple periodicity constraints. Consequently, each authorization is defined as being associated with a temporal constraint which takes care of the time during which the authorization exists.

Their authorization model is defined as follows:

- An "authorization" is defined like in Jajodia's work [42] – a 5-tuple (s, o, m, pn, g) where s, g are users, o is an object, m is an access mode and pn is either $+$ or $-$.
- A "periodic authorization" is given by a 4-tuple $([begin, end], P, auth)$ where "$begin$" and "end" signify time points, P is a periodic expression and "$auth$" is an authorization.
- "Derivation rules" are inference mechanisms used to come up with new authorizations. They are given by $([begin, end], P, A \langle op \rangle A')$ where "$begin$" and "end" are as above, A is an authorization, $\langle op \rangle$ is an operator which is one of WHENEVER, ASLONGAS or UPON with the usual meanings and A' is a boolean expression of authorizations.
- A "temporal authorization base" is a set of periodic authorizations and derivation rules.

Given a temporal authorization base, they illustrate how various authorizations can be derived from them. This generates the possibility of the set of derived authorizations derived not being unique and being conflicting with each other. They then present an algorithm which takes a temporal authorization base as input and rejects it if the set of derived authorizations is not unique.

Finally, they show how to model access control using a temporal authorization base and discuss implementation issues.

3.4 Generalized Authorization Problem

The work presented in this report in primarily based on the work by Schwoon et al. [15]. They formalize the concept of *Generalized Authorization Problem (GAP)* and show how this framework can be used to address issues such as privacy, recency, validity and trust. In the GAP framework the certificates are labeled with weights that are drawn from a bounded idempotent semiring. Two operators $\otimes$ and $\oplus$ of the semiring are defined which is used to calculate the value of a certificate chain and a set of certificate chains, respectively. The language that characterize the transition sequence for the PDS $\mathcal{P}$ and the automaton $\mathcal{A}$ for configuration set C is formalized. An algorithm to create an annotated automaton that is similar to the pre^* algorithm from [43] is also described. The annotation of weights against each certificate aids in answering several authorization related problems in which only that certificate chain which satisfies a specific criteria is required. A weighted SPKI/SDSI system $\mathcal{WSS}$ is a 3-tuple $(\mathcal{C}, \mathcal{S}, f)$, where $\mathcal{C}$ is a set of certs, $\mathcal{S} = (D, \oplus, \otimes, 0, 1)$ is a bounded idempotent semiring, and $f : \mathcal{C} \rightarrow D$ assigns weights to the certs in $\mathcal{C}$. The function f is extended to certificate chain as: $c_k \circ c_{k-1} \circ \cdots \circ c_1$, $f(c_k \circ c_{k-1} \circ \cdots \circ c_1)$ is defined as $f(c_1) \otimes \cdots \otimes f(c_{k-1}) \otimes f(c_k)$. A request r is a tuple (K, R, T), where K is the public key of the requester, R is the public key of the resource and T is the authorization requested. The framework of Generalized Authorization problem is formalized in Definition 14.

Definition 14. Given a weighted SPKI/SDSI system $\mathcal{WSS} = (\mathcal{C}, \mathcal{S}, f)$ and a request $r = (K', R, T'), proof(\mathcal{C}, r)$ denotes the set of certificate chains that prove that request r can be fulfilled. Formally, $proof(\mathcal{C}, r)$ is the set of certificate chain $c_k \circ \cdots \circ c_1$ not containing any useless certificates such that:

$$(c_k \circ \ldots c_1)(R\square) \in \{K'\square, K'\blacksquare\}$$

The generalized authorization problem (GAP) asks the following two questions: (1) Is $proof(\mathcal{C}, r)$ non-empty? (2) If $proof(\mathcal{C}, r)$ is non-empty, then find the following two quantities:

- $\delta = \oplus\{f(cc) \in proof(\mathcal{C}, r)\}$, cc is a certificate chain;
- a witness set of certificate chains $\omega \subseteq proof(\mathcal{C}, r)$ such that $\oplus_{cc \in \omega} f(cc) = \delta$.

It is shown, using Comprehensive Authorization Problem (CAP) framework, that associating WS1S formula to WPDS can provide further flexibility in answering additional authorization questions. In particular, the weighted domain in CAP consists of a regular language, represented as finite state automata, that is associated with each rule of WPDS (SPKI/SDSI certificate, respectively). The association of regular language based constraints, expressed as WS1S formula, allows definition of constraints that are used in the domain of access control. The expressiveness of the constraints expressed as WS1S formula is demonstrated using time constraints, represented as interval on an abstract domain. This representation allows authorization questions that can be more generic than those handled by GAP framework.

3.5 Logic and Access Control

Linear time logic (LTL) is a powerful query language for asking generalized reachability questions about a PDS. Jha et al. in [17] provide a method of associating LTL formula with configurations of PDS. Given a set of atomic propositions AP and a LTL formula ϕ over AP model-checking for LTL formula ϕ on PDS $\mathcal{P} = (P, \Gamma, \Delta)$ is done by associating using a labeling function Ω that associates each surface configuration $\langle p, \gamma \rangle$ with with a set of atomic formula $\Sigma = 2^{AP}$ as $\Omega : (P \times \Gamma) \rightarrow \Sigma$. Therefore, the set of atomic propositions that hold at a configuration $\langle q, \gamma w \rangle$ is given by $\Omega(\langle q, \gamma \rangle)$. The model checking problem is: "Given a configuration c of $\mathcal{P}$ and an LTL formula ϕ, determine whether c satisfies ϕ".

The model-checking problem is solved using a *Büchi pushdown system* which is a product of PDS and a Büchi automaton for which the acceptance condition is PDS augmented with Büchi acceptance condition. The Büchi pushdown system is defined as $\mathcal{BP} = ((P \times Q), \Gamma, \Delta', G)$, where

- $\langle (p, q), \gamma \rangle \hookrightarrow \langle (p', q'), w \rangle \in \Delta'$ if $\langle p\gamma \rangle \hookrightarrow \langle p'w \rangle$, $q \xrightarrow{\sigma} q'$, $\sigma \subseteq \Omega(\langle p, \gamma \rangle)$.
- $(p, q) \in G$ if $q \in F$.

Given a Büchi pushdown system, the accepting-run problem is the problem of answering the question "Is there an accepting run starting from the configuration $\langle (p, q), \gamma \rangle$ – i.e., a run that visits infinitely often the configurations with control locations in G?".

LTL model checking answers the following question about a PDS $\mathcal{P} = (P, \Gamma, \Delta)$ and labeling function Ω: "Does a configuration $c = \langle p, \gamma \rangle$ satisfy ϕ?. LTL model-checking can be reduced to the accepting run problem as follows:

- Construct the Büchi automaton $\mathcal{B} = (\Sigma, Q, \delta, q_0, F)$ corresponding to $\neg\phi$.
- Compute $\mathcal{BP}$ as the product of the Büchi automaton $\mathcal{B} = (\Sigma, Q, \delta, q_0, F)$ and the PDS $\mathcal{P} = (P, \Gamma, \Delta)$ with respect to the labeling function Ω.
- A configuration of $\langle p, w \rangle$ violates $\neg\phi$ iff there is an accepting run in $\mathcal{BP}$ starting from the configuration $\langle (p, q_0), \gamma \rangle$.

This result is of importance as the expressiveness of this formalism is similar to the one discussed in this thesis. The difference is in the approach presented here and the one developed in the work presented in this thesis is in the ease of definition of constraints using WS1S.

Glasgow et al. in [44] develop formal methods to reason about aspects of secrecy/security, integrity and accessibility. These are defined in terms of knowledge, permission and obligation as follows:

- **Secrecy:** If a subject (user or process) knows a formula, then the subject must be permitted to know that formula.

- **Integrity:** If a subject is obligated to know a formula, then it will eventually know it.
- **Accessibility:** If a subject is permitted to know a formula, then there should exist a means of finding it out.

The formal notation that is developed is called "security logic" and it is built in three steps. In the first step a simple modal logic with a "knowledge" modality is considered. The semantics considered is that of "possible worlds semantics" modeled through Kripke structures. In the second step, the authors consider the addition of time to the logic with knowledge above as the concept of integrity is tied to time. The authors choose the branching interpretation of time (i.e., not linear interpretation) in order to conveniently talk about a world and "all" its possible future worlds. Consequently, the semantics is extended where the underlying models talk about a branching tree of all possible paths and temporal operators to quantify over these paths is added to the logic.

In the third and final step (which leads to the full security logic) the authors extend the above logic by adding the deontic logic operators of "permission" and "obligation". For the purpose of defining semantics, models are extended to possess "permission sets" and "obligation sets" attached to subjects and these sets of formulas are true whenever the corresponding subject is permitted or obligated to know them, respectively.

At each of the three steps, the authors provide an axiom system and finally show that the axiom system for security logic is sound. Later, they illustrate how security policies can be defined in security logic. A point to be noted is that the security policies considered are static (i.e., they are not use-defined). This restriction is because permission set is attached to a subject and not to the system state. The above concept is illustrated by defining a multi-level secrecy policy in the logic and in the process, they also develop concepts for combining and composing policies. Two notions of combining policies are considered:

- **Compatibility:** A way of combining two policies such that the resulting policy satisfies both of the component policies and is still secure with respect to the logic.
- **Composability:** This deals with combining two different systems, each of which satisfies the same policy.

Finally, the authors illustrate how security logic can be used to reason about systems by modeling an information flow instance of multilevel secrecy, an integrity policy, two access control instances of multilevel secrecy, and an access-control based policy embodying both secrecy and integrity.

Chapter 4

Modeling Constraints using WS1S

4.1 Introduction

In the framework presented here the constraints are defined as a problem of deciding the membership in a regular language. For example, a rule "$\langle r_k \rangle\, [\mathcal{L}_m]$" denotes that the rule r_k is valid only if membership in the regular language $\mathcal{L}_m$ is guaranteed. Given a set of rules $\mathcal{R}$ the restriction and extension on the constraint is equivalent to intersection and union on the language associated with the rule. Given a WS1S formula the set of satisfying interpretations of the formula can be represented by a finite state automata. Hence, WS1S formula has been used to define and represent constraints. It is shown how a variety of constraints can be modeled using WS1S. The constraints that needs to be modeled is described informally followed by the equivalent WS1S formula. The constraints defined in this chapter have been validated using MONA [19, 20], therefore the WS1S formula for each constraint is also formulated in the input language that MONA accepts.

The fundamental premise that on which constraints is modeled using WS1S is based on [14] in which Büchi showed that every word in L_φ ,which is the language corresponding to the WS1S formula φ, has an interpretation for the free variables in φ under which φ evaluates to true. Every interpretation which makes φ true is represented by some word in L_φ. φ is satisfiable iff there is some interpretation which makes it true iff L_φ is nonempty. The language L_φ is defined over the alphabet $\{0,1\}^m$ where m is the number of free variables in φ. Given a WS1S formula φ an automata that accepts L_φ can be constructed inductively and the set of satisfying interpretations of a subformula is represented by a finite-state automaton. WS1S thus acts as a notation for regular languages, just as regular expressions do.

4.2 Periodicity Temporal Constraints

The example periodicity expression given in [7] is taken in which the temporal expression is formed by a periodic expression (for example, $9A.M.\ to\ 1\ P.M.\ on\ Working-days$, identifying the periods from 9 A.M. to 1 P.M. in all days excluding weekends and vacations), and a temporal interval bounding the scope of the periodic expression (example, $[2/1997, 8/1997]$, restricts the preceding periods to those between *February* and *August* of 1997). Such expressions are converted into an interval range and periodicity as $([t_1, t_2], p)$. The time range encodes both the time and date restrictions. The WS1S formula is given in Figure 4.1 where $is\_ValidTime$ is a predicate that evaluates that value of the time and periodicity values provided as input against the constraints specified, $Imax$, $Imin$ and p specifies the maximum and minimum time ranges and the periodicity of the constraint, respectively. These three parameters together define the periodicity temporal constraint. The parameters t and d represent the time and periodicity values, respectively. A temporal constraints is defined such that it consists of a

$$
\begin{array}{l}
is\_ValidTime(Imax, Imin, p, t, d) \\
\{ \\
(t \geq min(I) \wedge t \leq Imax \wedge \\
((is\_monday(p) \wedge is\_monday(d)) \vee \\
(is\_tuesday(p) \wedge is\_tuesday(d)) \vee \\
(is\_wednesday(p) \wedge is\_wednesday(d)) \vee \\
(is\_thursday(p) \wedge is\_thursday(d)) \vee \\
(is\_friday(p) \wedge is\_friday(d)) \vee \\
(is\_saturday(p) \wedge is\_saturday(d)) \vee \\
(is\_sunday(p) \wedge is\_sunday(d)))) \\
\}
\end{array}
$$

Figure 4.1: Formula for Periodicity Temporal Constraints

time variables $x, y, \ldots$ and an interval I which a convex subset of $\mathbb{N}$. As in [45] an interval I may be open, half open, or closed; bounded or unbounded. An interval associated to time variable is defined to be of the form $[a, b], [a, \infty), (a, b], (a, b)$ or (a, ∞), where $a \leq b$ and $a, b \in \mathbb{N}$. In the above definition a, b are called the left end-point and right end-point of I respectively.

In this model an interval is an ordered pair of points with the first point less than the second. If the left end-point of an interval I is represented by $l(I)$ and $r(I)$ represents right end-point relations between intervals I and J discussed in [46] can be defined in Table 4.1.

Interval Relation	**Equivalent Relations on End-points**
$t < s$	$(t+ < s-)$
$t = s$	$(t- = s-)\&(t+ = s+))$
t overlaps s	$(t- < s-)\&(t+ > s-)\&(t+ < s+)$
t meets s	$(t+ = s-)$
t during s	$((t- > s-)\&(t+ \leq s+))\|((t- \geq s-)\&(t+ < s+))$

Table 4.1: Interval Relations

```
#----Days of week-------
#monday | 0 | 0000 |
#tuesday | 1 | 0001 |
#wednesday | 2 | 0010 |
#thursday | 3 | 0011 |
#friday | 4 | 0100 |
#saturday | 5 | 0101 |
#sunday | 6 | 0110 |
#
macro is_monday(var1 p) =
p notin bit0 & p notin bit1 & p notin bit2 & p notin bit3 &
p notin bit4 & p notin bit5 & p notin bit6;
macro is_tuesday(var1 p) =
p in bit0 & p notin bit1 & p notin bit2 & p notin bit3 &
p notin bit4 & p notin bit5 & p notin bit6;
macro is_wednesday(var1 p) =
p notin bit0 & p in bit1 & p notin bit2 & p notin bit3 &
p notin bit4 & p notin bit5 & p notin bit6;
macro is_thursday(var1 p) =
p in bit0 & p in bit1 & p notin bit2 &
p notin bit3 & p notin bit4 & p notin bit5 & p notin bit6;
macro is_friday(var1 p) =
p notin bit0 & p notin bit1 & p in bit2 & p notin bit3 &
p notin bit4 & p notin bit5 & p notin bit6;
macro is_saturday(var1 p) =
p in bit0 & p notin bit1 & p in bit2 & p notin bit3 &
p notin bit4 & p notin bit5 & p notin bit6;
macro is_sunday(var1 p) =
p notin bit0 & p in bit1 & p in bit2 & p notin bit3 &
p notin bit4 & p notin bit5 & p notin bit6;
pred isValidTime(var2 Interval,
var1 periodicity,
var1 time,
var1 day) =
(time >= min(Interval) & time <= max(Interval) &
((is_monday(periodicity) & is_monday(day)) |
(is_tuesday(periodicity) & is_tuesday(day)) |
(is_wednesday(periodicity) & is_wednesday(day)) |
(is_thursday(periodicity) & is_thursday(day)) |
(is_friday(periodicity) & is_friday(day)) |
(is_saturday(periodicity) & is_saturday(day)) |
(is_sunday(periodicity) & is_sunday(day))));
```

Figure 4.2: The MONA code for WS1S formula in Figure 4.1

4.3 Attribute Based Constraints

An attribute based constraints is based on the attributes of the entities participating in access control system. Defining WS1S formula constraints based on attributes of entities is straight forward as it consists of comparison of the attribute against a value or a set of values.

Consider the case in which an attribute named 'experience' is used in the constraints so that the constraint is defined such that the value of this attribute is greater than or equal to some predefined constant value. This case can be extended so that the constraints contains more than one attribute conjoined together using conjunction, disjunction and negation operators. It is also possible to include universal quantification and existential operators in the definition of attribute based constraints so that the constraints can be defined on a group.

A hypothetical scenario that can be analyzed is a constraint on a group which restricts the group's access to a resource only if the group has at least one member whose 'experience' attribute has value greater than or equal to 25. This constraint can be represented as WS1S formula as given in Figure 4.3. In this formula $isAttribute\_Exp$ is predicate that evaluates true if x has attribute 'Experience' has value greater than or equal to '25'.

$$(\exists x : x \in X \wedge isAttribute\_Exp(x) \wedge x >= 25)$$

Figure 4.3: Formula for Attribute Based Constraints

```
#Experience | 7 | 0111 |

#The 'Experience' attribute is encoded as
pred isAttribute_Exp(var1 x) =
x in bit0 & x notin bit1 & x in bit2 &
x in bit3 & x notin bit4 & x >= 25;

pred matchAttribute(var2 SetOfAttributes) =
ex1 x: x in SetOfAttributes & isAttribute_Exp(x);
```

Figure 4.4: The MONA code for WS1S formula in Figure 4.3

4.4 Usage Based Constraints

The Usage Based Constraints enforces access control restriction based on usage count of the resource. This usage count can be specific to a user or it is global for the resource. For example, a simple access control policy over a room count context specifies that "a regular user can enter room C only if the number of regular users in C is less than the stipulated limit." We can express the formula corresponding to this policy with the following subset of Σ: $\{C_{max}, C^d_{max}, request\_entry\_C, allow\_entry\_C\}$ where

C_{max} is the event that the maximum occupancy has been reached, C^d_{max} is the dual of the C_{max} event which represents the fact that the current occupancy is less than the maximum limit. The event $request\_entry\_C$ is generated in the system when a request for entry into room C is made by the user and the event $allow\_entry\_C$ is generated when the user's request for entry into room C is granted.

$$
\begin{gathered}
\forall x, \forall y \\
(C^d_{max}(x) \wedge (x \leq y) \wedge request - entry - C(y) \wedge \\
(\neg \exists z((x \leq z) \wedge (z \leq y)) \wedge (C_{max}(z)) \Rightarrow \\
\exists w((y < w) \wedge allow - entry - C(w))) \\
\forall w \\
(allow - entry - C(w) \Rightarrow \\
\exists x, \exists y \\
(C^d_{max}(x) \wedge (x \leq y) \wedge request - entry - C(y) \wedge \\
(\neg \exists z((x \leq z) \wedge (z \leq y)) \wedge (C_{max}(z)) \wedge (y < w)))
\end{gathered}
$$

Figure 4.5: Formula enforcing usage constraint

Formula corresponding to the above policy, which regulates access of a regular user in room C, is shown gin Figure 4.5. Semantics of the policy can be understood from the words that satisfy it. The policy specifically rules out an occurrence of C_{max} (represented by z) between the last seen C^d_{max} (represented by x) and $request\_entry\_C$ (represented by y), if the request is to be followed by an $allow\_entry\_C$ in a valid word. In English, the policy formula means that entry to the room is allowed *if and only if* there was a request for entry and the relevant context (C^d_{max} in this case) already held. Exact formulation of such rules may differ from one policy to another. The MONA code corresponding to the WS1S formula in Figure 4.5 is given in Figure 4.6.

4.5 Spatial Constraints

Spatial constraints are primarily defined for granting access to a facility, which consists of spaces, to be protected. The topology of the facility is described in terms of rooms and their neighborhood relationships. Consider the example facility in Figure 4.7, which comprises of five rooms, viz. A, B, C, D, and outside of the facility - W. The specification of the topology is elucidated in Table 4.2. The access to a room is gained through the door(s) associated with the room. The decision on allowing or denying access to a room is taken when an access request is made by the user at the door corresponding to the room to which access is required.

In a normal scenario the facility administrator divides all the users into different classes or roles, and composes policies for each of these roles. First, this requires defining of contexts of interest using

```
pred request_into_C(var1 p, var1 q) =
q = p + 1 &
p in bit0 & p notin bit1 &
p notin bit2 & p notin bit3;

pred allow_into_C(var1 p, var1 q) =
q = p + 1 &
p notin bit0 & p in bit1 &
p notin bit2 & p notin bit3;

pred C_max(var1 p, var1 q) =
q = p + 1 &
p notin bit0 & p notin bit1 &
p in bit2 & p notin bit3;

pred C'_max(var1 p, var1 q) =
q = p + 1 &
p notin bit0 & p notin bit1 &
p notin bit2 & p in bit3;

pred can_enter_C(var1 p, var1 q) =
all1 r,s,s': C'_max(r, s) & request_into_C(s,s') &
p <= r & ((~ex1 t : (s <= t) & (t <= s') &
C_max(s', t) ) => ex1 s'' : allow_into_C(s,s') &
s' <= s'') =>
all1 u: allow_into_C(s', u) =>
ex1 x, y: C_max(u, x) & (x <= y) &
request_into_C(x, y) &
(~ex1 z : (x <= z) & (z <= y) &
C_max(y, z) & (y <= z));
```

Figure 4.6: The MONA code for WS1S formula in Figure 4.5

which the policies are framed. Contexts are defined in terms of events. An *event* is an occurrence or a happening of significance to one or more policies. The set of events includes *user actions* like requests for services, *application actions* like grant/denial of services, and *context events* that are constructed with the help of application actions and/or user actions and/or other context events. For every event representing some context, there is a *dual event* that represents the absence of the same context. Hence, for context events, either the event or its dual is always true. Henceforth, we use the terms *context* and *context event* interchangeably. In our notation, the dual of a context Z is represented by Z^d. To sum up, the set of events represents all the actions that occur in the system. *Behavior* of the system is modeled as a sequence of events, in the order in which they occur when the access control application is being executed.

Consider a generic policy where access to a room, say *RoomA*, is allowed only in presence of relevant context, say *Z*. *Z* can stand for the fact that occupancy is less than 10, or that a security guard is

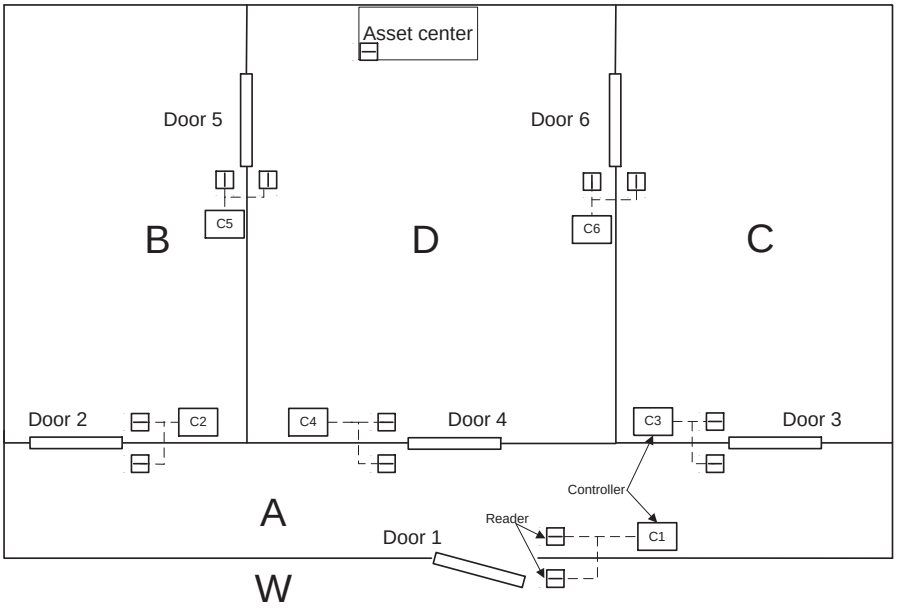

Figure 4.7: Example facility layout

```
# Input topology - List of rooms
rooms :  A,B,C,D,W;
# Input topology - Neighborhood
neighbor A: C,B,D,W;
neighbor B: A,D;
neighbor C: A,D;
neighbor D: A,B,C;
neighbor W: A;
```

Table 4.2: Facility topology specification

already present, or a conjunction of both. Here, we have four events, viz. *request-for-RoomA*, *allow-into-RoomA*, Z, and Z_D. The generic constraint may be written as: `Can enter RoomA on context Z`; the corresponding WS1S formula being:

$\forall t, \exists t' : Z(t) \wedge request - for - RoomA(t') \wedge \neg\exists t_D : Z_D(t_D) \wedge t < t_D \wedge t_D < t' \Rightarrow \exists t'' : allow - into - RoomA(t'') \wedge t'' = t'+1 \wedge \forall t'' : allow - into - Room(t'') \Rightarrow \exists t, t' : Z(t) \wedge request - for - RoomA(t') \wedge \neg\exists t_D : Z_D(t_D) \wedge t < t_D \wedge t_D < t' \wedge t' = t'' - 1$

which, in English means for all time instants, access is granted at this instant **if and only if** relevant context was already present, and an access request was made in the immediately preceding instant. Automaton for this policy is shown in 4.8. A detailed description of constraint for a sample facility is given in the Appendix section.

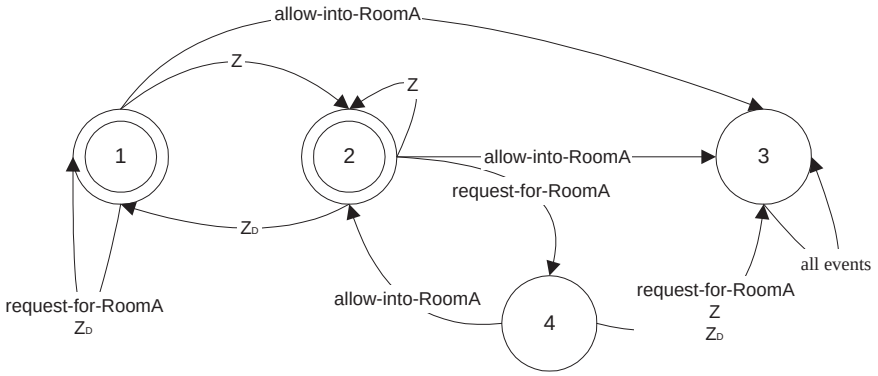

Figure 4.8: FSA for access control policy

4.6 Negative Constraints

All the constraints described above can be formulated into their corresponding negative constraints by taking a negation ($\neg$) of the constraint. Thus definition of negative constraint using WS1S is trivial.

As shown in the above sections, WS1S provides a very flexible mechanism for the definition of the constraints. These constraints when coupled with SPKI/SDSI validity condition provides a framework for the specification of PKI based infrastructure for access control in decentralized scenario. In the following section Weighted Pushdown System (WPDS), which is a known and well studied formal model, is extended so that WS1S constraints can be included into the formal framework. We call this framework as Augmented WPDS (AWPDS).

Chapter 5

Modification to WPDS

5.1 Introduction

In the previous chapter it was demonstrated that flexible constraints can be modeled using WS1S formulas. If these constraints can be associated to the rules of WPDS then a more flexible system can be obtained. The WPDS system allows association of weights to each rule of the PDS. These weights which are taken from the domain of bounded idempotent semiring brings in restriction to each rule. The problem that is addressed in this chapter is the association of the rules of PDS with WS1S based constraints. To use the formulation of WPDS it is required that the set of constraints that are associated to the rule be a bounded idempotent semiring. It is known that WS1S formula enables convenient representation but not execution. Therefore, the set of *satisfying interpretations* of a WS1S formula which denotes the constraints is represented by a finite-state automaton. The automaton for a formula can be calculated by a simple induction scheme, where logical connectives correspond to classic automata-theoretic operations such as product and subset constructions. Validity and unsatisfiability of the formula can be determined and satisfying examples and counter-examples can be constructed by analyzing the associated automaton [18].

5.2 Augmentation to Weighted Pushdown System (AWPDS)

In WPDS weights are allocated to each rule of PDS. The weights allocated are values from a domain which is a bounded idempotent semiring. To associate generic constraint that define membership in a regular language the WPDS is augmented such that each rule can be associated with WS1S formula which is a succinct representation of a regular language of interest. Association of WS1S formula to each rule does not extend WPDS into a new model but only defines a new idempotent semiring domain. This augmentation aids in increasing the expressibility of the WPDS so that more powerful constraints

can be associated.

5.2.1 Augmented Weighted Pushdown System (AWPDS)

As described earlier, the augmentation to WPDS is in terms of WS1S constraints that can be associated against each rule. The augmentation is as defined below.

Definition 15. Augmented Weighted Pushdown System (AWPDS) $\stackrel{def}{=}$ An AWPDS is a tuple $\mathcal{A} = (\mathcal{W}, \mathcal{F}, h)$ such that $\mathcal{W} = (\mathcal{P}, \mathcal{S} \times \mathcal{M}, g)$ is a weighted pushdown system, with $\mathcal{P} = (P,\ \Gamma, \Delta)$ in which P is a set of control locations, Γ is set of stack alphabet, and Δ is the set of rules. $\mathcal{F}$ is the set of WS1S formula and $h : \Delta \rightarrow \mathcal{F}$ is a function that assigns a value from $\mathcal{F}$ to each rule of $\mathcal{P}$. $\mathcal{S} = (D, \oplus, \otimes, 0, 1)$ and $\mathcal{M} = (M, \cup, \cap, \emptyset, \mathbb{M}_U)$ are two idempotent semirings. $\mathbb{M}$ is a set m of Finite State Automata (FSA) such that for all $f \in \{h(\Delta) \cup f_c\}$ in which f_c is the formula corresponding to the constraints that the AWPDS will be evaluated for and m is the FSA corresponding to all satisfying interpretations of f and $\mathtt{M}_U = \cup_{x \in \mathbb{M}} x$. The set M is formed from $\mathtt{M}$ such that $M = \{x | y \in 2^{\mathbb{M}} \wedge x = \cup_{a \in y} a\}$, i.e. the elements of M are the union of subsets of the automaton in $\mathbb{M}$. The function g maps the rules of $\mathcal{P}$ to the elements from D and M as $g : \Delta \rightarrow (D \times M)$.

Theorem 1. $\mathcal{M} = (M, \cup, \cap, \emptyset, \mathbb{M}_U)$ is a bounded idempotent semiring.

Proof. The set M along with operators $\cup$ (union), $\cap$ (intersection) and elements $\emptyset \in M$ and $\mathbb{M}_U \in M$ satisfies the following properties:

1. $(M, \cup)$ is a commutative monoid with $\emptyset$ as its neutral element, and where $\cup$ is idempotent as $\forall\ \mathtt{m} \in M,\ \mathtt{m} \cup \emptyset = \mathtt{m}$.
2. $(M, \cap)$ is a monoid with the neutral element $\mathbb{M}_U$ as $\forall\ \mathtt{m} \in M,\ \mathtt{m} \cap \mathtt{M}_U = \mathtt{M}_U \cap \mathtt{m} = \mathtt{m}$
3. By the property of regular languages $\cap$ distributes over $\cup$, therefore, for all $\mathtt{m}_1, \mathtt{m}_2, \mathtt{m}_3 \in M$ we have, $\mathtt{m}_1 \cap (\mathtt{m}_2 \cup \mathtt{m}_3) = (\mathtt{m}_1 \cap \mathtt{m}_2) \cup (\mathtt{m}_1 \cap \mathtt{m}_3)$ and $(\mathtt{m}_1 \cap \mathtt{m}_2) \cup \mathtt{m}_3 = (\mathtt{m}_1 \cap \mathtt{m}_3) \cup (\mathtt{m}_2 \cap \mathtt{m}_3)$.
4. $\emptyset$ is an annihilator with respect to $\cap$, i.e., for all $\mathtt{m} \in M$, $\mathtt{m} \cap \emptyset = \emptyset = \emptyset \cap \mathtt{m}$.
5. In the partial order $\sqsubseteq$ defined by: for all $\mathtt{m}_1, \mathtt{m}_2 \in \mathbb{M}$, $\mathtt{m}_1 \sqsubseteq \mathtt{m}_2$ iff $\mathtt{m}_1 \cup \mathtt{m}_2 = \mathtt{m}_1$, there are no infinite descending chains as $\mathtt{M}_U \in M$ forms the infimum.

Therefore, $\mathcal{M}$ is a bounded idempotent semiring. $\square$

Theorem 2. Let $S_1 = (D_1, \oplus, \otimes, 0, 1)$ and $S_2 = (D_2, \boxplus, \boxtimes, \bot, \top)$ be two idempotent semirings. Their cartesian product $S_1 \times S_2 = ((D_1 \times D_2), +, *, \{0, \bot\}, \{1, \top\})$ is also an idempotent semiring.

Proof. The set $(D_1 \times D_2)$ along with operators $+$ and $*$ defined on elements $(p1, p2) \in (D_1 \times D_2)$ and $(q1, q2) \in (D_1 \times D_2)$ as $(p1, p2) + (q1, q2) = (p1 \oplus q1, p2 \boxplus q2)$ and $(p1, p2) * (q1, q2) = (p1 \otimes q1, p2 \boxtimes q2)$, respectively, and elements $(0, \bot) \in (D_1 \times D_2)$ and $(1, \top) \in (D_1 \times D_2)$ satisfies the following properties:

1. $((D_1 \times D_2), +)$ is a commutative monoid with $(0, \bot)$ as its neutral element, and where $+$ is idempotent as $\forall\ (p,q) \in (D_1 \times D_2)$, $(p,q) + (0,\bot) = (p \oplus 0, q \boxplus \bot) = (0,\bot) + (p,q) = (p,q)$.

2. $((D_1 \times D_2), *)$ is a monoid with the neutral element $(1, \top)$ as $\forall\ (p,q) \in (D_1 \times D_2)$, $(p,q) * (1,\top) = (p * 1, q * \top) = (1,\top) * (p,q) = (p,q)$

3. $*$ distributes over $+$ (Theorem 3).

4. $(0,\bot)$ is an annihilator with respect to $*$, i.e., for all $(p,q) \in (D_1 \times D_2)$, $(p,q) * (0,\bot) = (0,\bot) = (0,\bot) * (p,q)$.

5. Define a partial order between elements $(p1,p2), (q1,q2) \in (D_1 \times D_2)$ such that $(p1,p2) \sqsubseteq (q1,q2)$ iff $(p1 \sqsubseteq p2)$ and $(q1 \sqsubseteq q2)$. This partial order has no infinite descending chain as $(1, M)$ defines the infimum.

Therefore, $(D_1 \times D_2)$ is a bounded idempotent semiring. $\square$

Theorem 3. Let the set $(D_1 \times D_2)$ have operators $+$ and $*$ defined as $(p1,p2) \in (D_1 \times D_2)$ and $(q1,q2) \in (D_1 \times D_2)$ as $(p1,p2) + (q1,q2) = (p1 \oplus q1, p2 \boxplus q2)$ and $(p1,p2) * (q1,q2) = (p1 \otimes q1, p2 \boxtimes q2)$, respectively. Then, $*$ distributes over $+$.

Proof. Let $(p1,p2), (q1,q2), (r1,r2) \in (D_1 \times D_2)$

$(p1,p2) * ((q1,q2) + (r1,r2))$

$= (p1 \otimes (q1 \oplus r1), p2 \boxtimes (q2 \boxplus r2))$

$= (p1 \otimes q1 \oplus p1 \otimes r1), (p2 \boxtimes q2 \boxplus p2 \boxtimes r2)$

$= ((p1 \otimes q1), (p2 \boxtimes q2)) + ((p1 \otimes r1), (p2 \boxtimes r2))$

$= ((p1,p2) * (q1,q2)) + ((p1,p2) * (r1,r2))$

Therefore, $(p1,p2) * ((q1,q2) + (r1,r2)) = ((p1,p2) * (q1,q2)) + ((p1,p2) * (r1,r2))$

Similarly,

$((p1,p2) + (q1,q2)) * (r1,r2))$

$= ((p1 \oplus q1), (p2 \boxplus q2)) * (r1,r2)$

$= ((p1 \oplus q1) \otimes r1), ((p2 \boxplus q2) \boxtimes r2)$

$= ((p1 \otimes r1) \oplus (q1 \otimes r1)), ((p2 \boxtimes r2) \boxplus (q2 \boxtimes r2))$

$= ((p1 \otimes r1), (p2 \boxtimes r2)) + ((q1 \otimes r1), (q2 \boxtimes r2))$

$= ((p1,p2) * (r1,r2)) + ((q1,q2) * (r1,r2))$

$((p1,p2) + (q1,q2)) * (r1,r2)) = ((p1,p2) * (r1,r2)) + ((q1,q2) * (r1,r2))$

Hence, $*$ distributes over $+$. $\square$

Let $\sigma \in \Delta^*$ be a sequence of rules. Using g, it is possible to associate a value to σ such that if $\sigma = [r_1, \ldots, r_k]$, then define $v(\sigma) := g_{\mathcal{S}}(r_1) \otimes \cdots \otimes g_{\mathcal{S}}(r_k)$ and $u(\sigma) := g_{\mathcal{M}}(r_1) \cap \cdots \cap g_{\mathcal{M}}(r_k)$, where $g_{\mathcal{S}}$ and $g_{\mathcal{M}}$ are projections on elements of D and M in $(D \times M)$, respectively. For any two configurations

c and c' of $\mathcal{P}$, let path(c, c') denote the set of all rule sequences $[r_1, \ldots, r_k]$ that transform c into c', i.e., $c \overset{\langle r_1 \rangle}{\Longrightarrow}, \ldots, \overset{\langle r_k \rangle}{\Longrightarrow} c'$.

Definition 16. Given a AWPDS $\mathcal{A} = (\mathcal{W}, \mathcal{F}, h)$ such that $\mathcal{W} = (\mathcal{P}, \mathcal{S} \times \mathcal{M}, g)$ is a weighted pushdown system, with $\mathcal{P} = (P, \Gamma, \Delta)$, and a regular set of configurations $C \subseteq P \times \Gamma^*$, the Generalized Pushdown Reachability (GPR) problem is updated so that for each $c \in P \times \Gamma^*$:

- $\delta(c) := \bigoplus \{v(\sigma) \,|\sigma \in path(c, c'), c' \in C\}$;
- $\psi(c) := \bigcup \{u(\sigma) \,|\sigma \in path(c, c'), c' \in C\}$;
- a witness set of paths $\omega(c) \subseteq \bigcup_{c' \in C} path(c, c')$ such that $\bigoplus_{\sigma \in \omega(c)} v(\sigma) = \delta(c)$ and $\bigcup_{\sigma \in \omega(c)} u(\sigma) = \psi(c)$.

5.3 Algorithm for GPR problem in AWPDS

This section describes the algorithm that is a modification on the algorithm presented in [16]. The modification specifically pertains to the WS1S constraints associated to each rule. The algorithm is given for predecessor version of GPR given in Definition 16.

5.3.1 Overview

The input is an Augmented Weighted Pushdown System (AWPDS) $\mathcal{A} = (\mathcal{W}, \mathcal{F}, h)$ along with the constraint f_c that should be satisfied, where $\mathcal{W} = (\mathcal{P}, \mathcal{S} \times \mathcal{M}, g)$ is a weighted pushdown system, with $\mathcal{P} = (P, \Gamma, \Delta)$ and $\mathcal{S} = (D, \oplus, \otimes, 0, 1)$, $\mathcal{S} = (M, \cup, \cap, \bot, \top)$ and a regular set C of configuration. The output is a $pre^*(C)$ automaton from which $\delta(c)$ and $\psi(c)$ and a witness set of path $\omega(c)$ can be computed.

In general, there are infinitely many configurations in $pre^*(C)$ even if C itself is finite, therefore, an annotated finite automata is used for the purpose of computing $pre^*(C)$.

Definition 17. A $\mathcal{P}$-automaton is a quintuple $A = (Q, \Gamma, \eta, P, F)$ where $Q \supseteq P$ is a finite set of states, $\eta \subseteq Q \times \Gamma \times Q$ is the set of transitions, and $F \subseteq Q$ are the final states. The initial states of A are the control locations P. We say that a sequence of transitions $(p, \gamma_1, p_1), \ldots, (p_{n-1}, \gamma_n, q) \in \eta$ reads configuration $(p, \gamma_1, \ldots, \gamma_n)$ if $p, \ldots, p_{n-1}, q$ are arbitrary states. The sequence is accepting iff q is a final state. If c is a configuration of A, we denote by $\text{acc}_A(c)$ the set of all paths in A for c. The configuration c is accepted by A is $\text{acc}_A(c)$ is non-empty.

It is known from the discussions in [43] and [47] that the backward (resp. the forward) reachability closure of a regular set of configuration is regular. Given an automaton A that accepts the set C of regular configuration, it is possible to construct an automata that accepts the sets of all configurations

that are forward or backwards reachable from C. In the automaton constructed two additional labeling for the transitions of A are computed for the GPP. The first labeling introduced in [15], $l : \eta \rightarrow D$ assigns a weight from D to each automaton transition and allows to compute δ. The second labeling $m : \eta \rightarrow M$ assigns a finite state automata allows to compute final $pre^*(C)$ automaton. The WS1S formula is converted into an automaton representation for all the satisfying assignments of the WS1S formula for easier and intuitive computation of union and intersection operations.

5.3.2 Description

Let $A = (Q, \Gamma, \eta, P, F)$ be a $\mathcal{P}$-automaton accepting a set of configurations C. Initially, for all $t \in \eta$, let $l(t) := 1$ and $m(t) := FSA(\mathtt{f}_c)$, where $\mathtt{f}_c$ is the final constraint of the system against which the AWPDS is evaluated and $FSA(f)$ returns the FSA corresponding to all satisfying assignments of the WS1S formula f. The following is meant when a transition t should be updated with value $d \in D$ and $m \in M$: if t is not yet in η, add t to η and set $l(t) := l(t) \oplus d$ and set $m(t) := m(t) \bigcup m$.

For GPP, new transitions are added according the following saturation rules. These rules are an extension to the original set of rules given in [16]:
If $r := \langle p, \gamma \rangle \hookrightarrow \langle p', \omega \rangle$ is a rule, $t_1 \dots t_{|\omega|}$ a sequence that reads $\langle p, \omega \rangle$ and ends in state q, then:

- let d be $l(t_1) \otimes \cdots \otimes l(t_{|\omega|})$ and update (p, γ, q) with the value $g_D(r) \otimes d$
- let m be $m(t_1) \bigcap \cdots \bigcap m(t_{|\omega|})$ and update (p, γ, q) with the value $g_M(r) \bigcap m$.

The algorithm stops when further application of the saturation rule causes no further changes in A.

5.3.3 Algorithm

The algorithm for computing the automaton A given the configuration set C and AWPDS $\mathcal{A}$ is given in Algorithm 1. The algorithm is similar to the certificate chain discovery algorithm in [16] except for the following additions:

1. The transitions of input automaton corresponding to configuration C is updated with the final constraint. For all transactions t in the automaton corresponding to the initial configuration C set $m(t) = FSA(\mathtt{f}_c)$ where $\mathtt{f}_c$ is the WS1S formula corresponding to final constraint.
2. The transitions are updated so that both the $l(t)$ and $m(t)$ are updated. For each transition update the m.
3. The $UPDATE$ method is modified so that the transition's $m(t)$ is updated with new value.

The $\bigcup$ and $\bigcap$ operators are used to *extend* and *restrict* the constraint on the transition, respectively. Given two constraints the restrict operator evaluates to a constraint that is an intersection of the two

constraints resulting in 'tightening' of constraint. The extend operator 'relaxes' the constraint so that only one of them need to be valid. From a language theoretic point of view, the extend and restrict operators modify the language by taking union and intersection respectively.

5.3.4 The Connection Between SPKI/SDSI and Augmented Weighted Pushdown Systems

The connection between SPKI/SDSI and AWPDS is similar to one described in §2.2.3 with extra connection established for the function g which maps each certificate/rule of SPKI to a WS1S formula that defines generic constraint for each rule along with the weights defined in [15].

Example 1. The connection between SPKI/SDSI is illustrated using the certificate set $\mathcal{C}$ shown in Figure 5.1. The augmented weighted pushdown system $\mathcal{A}_\mathcal{C} = (\mathcal{W}_\mathcal{C}, \mathcal{F}_\mathcal{C}, h_\mathcal{C})$ $\mathcal{P}_\mathcal{C} = (\mathcal{K}_\mathcal{C}, \mathcal{I}_\mathcal{C} \cup \{\square, \blacksquare\}, \Delta_\mathcal{C})$, $\mathcal{S} = (\{T\}, \otimes, \oplus, \bot, \top)$, in which the $\otimes$ operators is defined so that $T \otimes \top = \top, T \otimes \bot = \bot$, and $\oplus$ is defined such that $T \oplus \top = \top, T \oplus \bot = T$. The certificates are associated with time range constraints. The set of all constraint is $\mathcal{F} = \{[10 \leq x \leq 15], [5 \leq x \leq 9], [5 \leq x \leq 20], [10 \leq x \leq 20]\}$. These time range constraint is mapped to the rules as $g = \{(r_1, (T, [10 \leq x \leq 15])), (r_2, (\top, [5 \leq x \leq 9])), (r_3, (\top, [10 \leq x \leq 15])), (r_4, (\top, [5 \leq x \leq 20])), (r_5, (\top, [10 \leq x \leq 20]))\}$.

The Generalized Authorization Problem (GAP) described in [16] is updated so that the authorization problem also includes the constraints associated with each certificate. A framework called Comprehensive Authorization Problem (CAP) is described. It is shown that the CAP framework solves the GAP and other extended problems.

5.4 Comprehensive Authorization Problems

Given a principal K, a set of name and authorization certificates $\mathcal{C}$, a resource R, and system constraint ξ the *Comprehensive Authorization Problem* (CAP) in the context of SPKI/SDSI certificate system addresses the questions – Is K authorized to access resources R given ξ?. In the framework of CAP the constraint ξ can be any kind of constraint that can be encoded using WS1S and hence manipulated using operations on a finite state automata. The AWPDS is used to answer the CAP and arrive at a solution. In the following it is demonstrated that several security-policy constraints like those described in [15] along with several others can be answered using CAP framework.

The CAP is a further generalization of the approach used for Generalized Authorization Problem (GAP) which uses weighted pushdown system (WPDS). The general approach to solving CAP is as follows:

- The initial automaton that accepts the configuration is constructed.

Algorithm 1 An algorithm for creating a extended Weighted Automaton for the GPP problem.

Input: an Augmented Weighted Pushdown System AWPDS $\mathcal{A} = (\mathcal{W}, \mathcal{F}, h)$ such that $\mathcal{W} = (\mathcal{P}, \mathcal{S} \times \mathcal{M}, g)$ is a weighted pushdown system, with $\mathcal{P} = (P, \Gamma, \Delta)$ in which Γ is the set of stack alphabet, Δ is the set of rules, $\mathcal{F}$ is a set of valid formula of WS1S and $h : \Delta \to \mathcal{F}$ is a function that assigns a value from $\mathcal{F}$ to each rule of Δ of P; and $\mathtt{f}_c$ which is the WS1S formula representing the constraint against which the rules needs to be evaluated. An initial automaton that accepts the configuration C.

Output: an automaton $A' = (Q, \Gamma, \eta, P, F)$, for $pre^*(C)$ with annotation functions $l : \eta \to D$ and $m : \eta \to M$.

procedure UPDATE(t, v, u)
  $\eta := \eta \cup \{t\}$
  $newLValue := l(t) \oplus v$
  $newMValue := m(t) \bigcup u$
  if $newLValue \neq l(t)$ or $newMValue \neq m(t)$ **then**
    $workSet := workSet \cup \{t\}$
    $l(t) := newLValue$
    $m(t) := newMValue$
  end if
end procedure

$\eta := \eta_0$; $workSet := \eta_0$;
for all $t \in \eta_0$ **do**
  $l(t) := 1$
  $m(t) := FSA(\mathtt{f}_c)$
end for
for all $r = \langle p, \gamma \rangle \hookrightarrow \langle p', \epsilon \rangle \in \Delta$ **do**
  UPDATE$((p, \gamma, p'), g_D(r), g_M(r))$
end for
while $workSet \neq \emptyset$ **do**
  remove some transition $t = (q, \gamma, q')$ from $workSet$;
  for all $r = \langle p_1, \gamma_1 \rangle \hookrightarrow \langle q, \gamma \rangle \in \Delta$ **do**
    UPDATE$((p_1, \gamma_1, q'), g_D(r) \otimes l(t), g_M(r) \bigcap m(t))$
  end for
  for all $r = \langle p_1, \gamma_1 \rangle \hookrightarrow \langle q, \gamma\gamma_2 \rangle \in \Delta$ **do**
    for all $t' = (q', \gamma_2, q'') \in \eta$ **do**
      UPDATE$((p_1, \gamma_1, q''), g_D(r) \otimes l(t) \otimes l(t'), g_M(r) \bigcap m(t) \bigcap m(t'))$
    end for
  end for
  for all $r = \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p', \gamma_2\gamma \rangle \in \Delta$ **do**
    if $t' = (p', \gamma_2, q) \in \eta$ **then**
      UPDATE$((p_1, \gamma_1, q'), g_D(r) \otimes l(t') \otimes l(t), g_M(r) \bigcap m(t') \bigcap m(t))$
    end if
  end for
end while
return $((Q, \Gamma, \eta, P, F), l, m)$

- A set of labeled SPKI/SDSI certificates with WS1S constraints associated with each one of them is first translated into to an augmented weighted pushdown system (AWPDS) using the connection demonstrated in Section §5.3.4.
- A shortest path problem on AWPDS is solved for finding solution to CAP.
  - A *pre** automaton is formed from the initial (basic) automaton by applying saturation rule to add new transitions.
  - Graph shortest path algorithm is used to find the shortest path from the key, for which authorization is being evaluated, to the final state of the automaton, such that the transition is annotated with the minimum (or maximum, as applicable) of the constraint specified.

$$r_1 := \langle K_r, \square \rangle \hookrightarrow \langle K_{uw}, \texttt{faculty}\blacksquare \rangle \, (T, [10, 15]) \tag{5.1}$$
$$r_2 := \langle K_{uw}, \texttt{faculty} \rangle \hookrightarrow \langle K_{ls}, \texttt{faculty} \rangle \, (\top, [5, 9]) \tag{5.2}$$
$$r_3 := \langle K_{ls}, \texttt{faculty} \rangle \hookrightarrow \langle K_{cs}, \texttt{faculty} \rangle \, (\top, [10, 15]) \tag{5.3}$$
$$r_4 := \langle K_{ls}, \texttt{faculty} \rangle \hookrightarrow \langle K_{bio}, \texttt{faculty} \rangle \, (\top, [5, 20]) \tag{5.4}$$
$$r_5 := \langle K_{cs}, \texttt{faculty} \rangle \hookrightarrow \langle K_{Bob}, \epsilon \rangle \, (\top, [10, 20]) \tag{5.5}$$

Figure 5.1: Example certificate list

Example 2. Consider the SPKI/SDSI certificate set given in Figure 5.1 which consists of five certificates each associated with temporal constraints given in the form $[t_1, t_2]$ which represents the constraint that the certificate r_i to which it is associated is valid only between the time range t_1 and t_2. The representation of temporal constraint is in form of interval in an abstract domain but the WS1S formula that is used during CAP as given below in which t1 and t2 are the values from the interval defined against the certificate:

```
var2 I where I={t1,...,t2};
var1 t;
#
pred Evaluate(var2 Interval, var1 time) =
(time in I);
#
Evaluate(I, t);
```

Suppose that *Bob* wants permission t from the resource owner K_r within the time units [10, 20]. The authorization in CAP framework finds solution to the following problem: "Is *Bob* authenticated by the key K_{Bob} allowed access to access to resource r identified by the key K_r ***between the time range***

10 and 20?" The answer to the question is determined by solving the GPP problem on AWPDS and configuration $C = \{\langle K_{Bob}, \square\rangle, \langle K_{Bob}, \blacksquare\rangle\}$.

The automaton that accepts the configuration $C = \{\langle K_{Bob}, \square\rangle, \langle K_{Bob}, \blacksquare\rangle\}$ is given in Figure 5.2. The final automaton output by the algorithm Algorithm 1 is given in Figure 5.3.

To provide an insight into the automaton that labels the transitions of the final automaton of Algorithm 1 the automaton for the transition $K_{uw} \overset{faculty}{\rightarrow} K_{Bob}$ is shown in Figure 5.4. Though the automaton looks complex the automaton is only used during the analysis phase. The MONA code that is used for performing union and intersection operations on the automaton is given in Figure 5.5 and Figure 5.6, respectively.

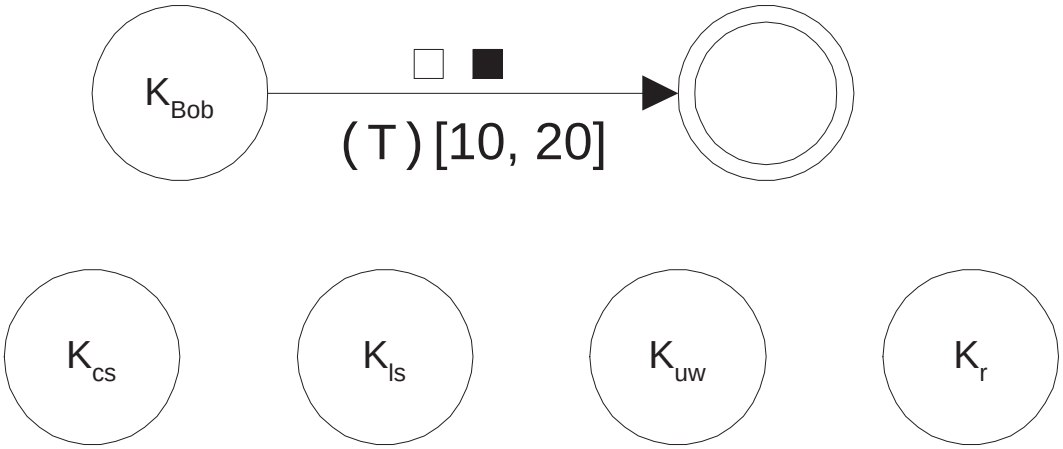

Figure 5.2: Initial automaton created by the algorithm

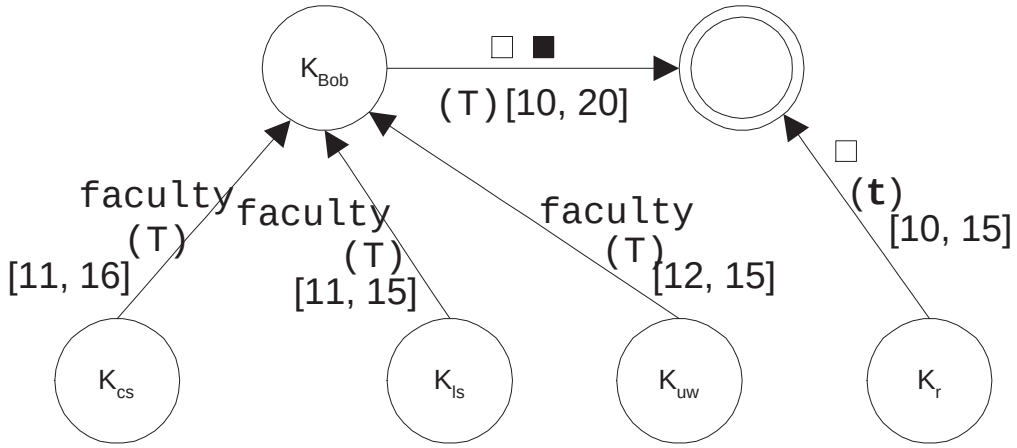

Figure 5.3: Automaton created by the algorithm

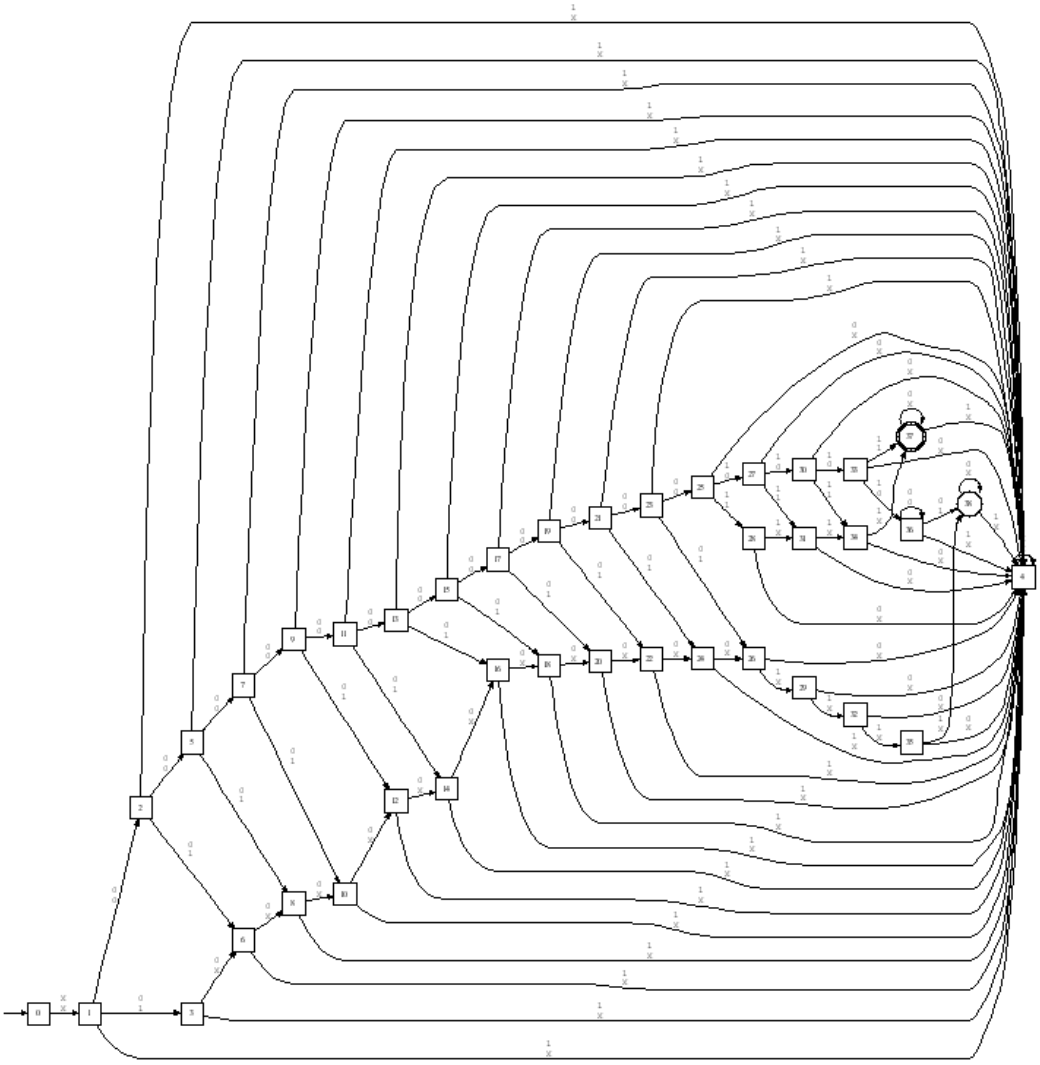

Figure 5.4: The automaton labeling the transition $K_{uw} \overset{faculty}{\rightarrow} K_{Bob}$

5.4.1 Power and Expressiveness of CAP

The main objective of augmenting WPDS and defining a CAP framework is to enable higher level of expressiveness for the validity constraints specified against a SPKI/SDSI certificates.

The previous sections described with examples how AWPDS and WS1S based constraints can solve authorization problem in which the constraints are defined as interval on abstract domain. This section provides details on the use of CAP framework for addressing the access control problems discussed in the literature including those addressed by Schwoon et al. in [15].

```
var2 I1 where I1={10,...,15};
var2 I2 where I2={10,...,20};
var2 R;
var1 max';
var1 min';

pred Extend(var2 P, Q, var1 x, y) =
R = P inter Q & x = max(R) & y = min(R);

Extend(I1, I2, max', min');
```

Figure 5.5: The MONA code for ***Extending*** constraints

```
var2 I1 where I1={11,...,15};
var2 I2 where I2={12,...,19};
var2 R;
var1 max';
var1 min';

pred Combine(var2 P, Q, var1 x, y) =
(empty(P inter Q) => (x = 0 & y = 0)) &
(~empty(P inter Q) => (x = max(P union Q) &
y = min(P union Q)));

Combine(I1, I2, max', min');
```

Figure 5.6: The MONA code for ***Combining*** constraints

5.4.1.1 Privacy-preserving certificate chains

The concept of privacy preserving certificate chains is described by Schwoon et al. in [15] and motivated by the example given in Figure 5.7.

In this example company X offers additional insurance to patients of a certain hospital H. $K_X\Box$ represents the service offered by company X. The principals corresponding to AIDS and internal-medicine clinics in hospital H are denoted by K_{H-AIDS} and K_{H-IM}. Alice is a patient in both the clinics.

Consider the case in which Alice wants to buy the insurance from the insurance company X providing the special service to hospital H. The certificate set given in 5.7 results in two chains $(r_4)\circ(r_2)\circ(r_1)$ and $(r_5)\circ(r_3)\circ(r_1)$ both of which is equal to $K_X\Box \rightarrow K_{Alice}\blacksquare$. However, the certificate chain $(r_4)\circ(r_2)\circ(r_1)$ reveals that Alice probably has AIDS, which is an information that Alice may not which to reveal to company X. Therefore, Alice would prefer to offer the certificate chain $(r_5) \circ (r_3) \circ (r_1)$ to company X which proves that she is authorized to buy additional insurance, but reveals the least amount of information about her.

Privacy can be modeled in the CAP framework using the following logic:

- non-overlapping intervals $[S]$ and $[I]$ corresponding to "sensitive" (S) and "insensitive" (I) information such that the $max([I]) < min([S])$.
- solve the CAP problem - "is there a certificate chain that only reveals insensitive information (corresponding to the interval $[I]$?"

The initial automaton, final automaton and the certificate chain for the example certificate set of Figure 5.7 is given in Figure 5.8, Figure 5.9 and Table 5.1 respectively. In this example, $[I] = [1, 5]$ and $[S] = [6, 10]$.

$$r_1 := \langle K_X, \square \rangle \hookrightarrow \langle K_H, \texttt{patient}\blacksquare \rangle \, [I] \tag{5.6}$$
$$r_2 := \langle K_H, \texttt{patient} \rangle \hookrightarrow \langle K_{H-AIDS}, \texttt{patient} \rangle \, [I] \tag{5.7}$$
$$r_3 := \langle K_H, \texttt{patient} \rangle \hookrightarrow \langle K_{H-IM}, \texttt{patient} \rangle \, [I] \tag{5.8}$$
$$r_4 := \langle K_{H-AIDS}, \texttt{patient} \rangle \hookrightarrow \langle K_{Alice}, \epsilon \rangle \, [S] \tag{5.9}$$
$$r_5 := \langle K_{H-IM}, \texttt{patient} \rangle \hookrightarrow \langle K_{Alice}, \epsilon \rangle \, [I] \tag{5.10}$$

An I associated to the certificate represents "insensitive" information. Similarly, "S" against the certificate denotes the fact that the certificate reveals sensitive information.

Figure 5.7: Example certificate set for Privacy preserving chain

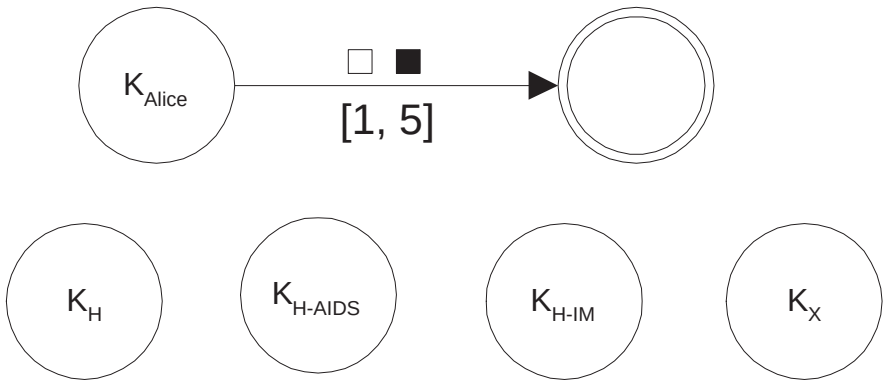

Figure 5.8: Initial automaton for privacy preserving certificate chain of 5.7

(State, Transition)	**Rule Applied**
$\langle K_X, \square \rangle$	$\langle K_X, \square \rangle \hookrightarrow \langle K_H, patient \blacksquare \rangle$
$\langle K_H, patient \blacksquare \rangle$	$\langle K_H, patient \rangle \hookrightarrow \langle K_{H-IM}, patient \rangle$
$\langle K_{H-IM}, patient \blacksquare \rangle$	$\langle K_{H-IM}, patient \rangle \hookrightarrow \langle K_{Alice}, \epsilon \rangle$
$\langle K_{Alice}, \blacksquare \rangle$	

Table 5.1: Example Privacy Certificate Chain

This CAP model can solve a variant of the privacy preserving problem in which multiple privacy or sensitivity levels can be associated to the certificate set and the CAP question can be modified to find a chain which only includes certificates below a certain level of privacy and sensitivity.

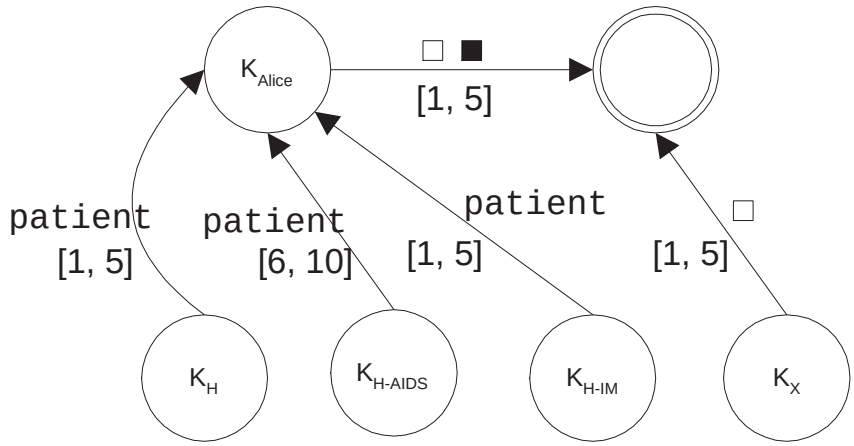

Figure 5.9: Final automaton for privacy preserving certificate chain of 5.7

5.4.2 Maximally-valid certificate chain

As in section §5.4.1.1 the definition of Maximally-valid certificate chain is described as follows in [15]: Let $V(c)$ be the expiration value of cert c, i.e., the cert c will expire at time $T_{current} + V(c)$, where $T_{current}$ is the current time. The expiration value of a certificate chain $c_k \circ c_{k-1} \circ \cdots \circ c_1$ is $min_{i=1}^{k} V(c)$. Suppose that Alice wants to login to host H. If Alice provides a certificate chain that is only valid for two minutes, then she will be logged off by the host after two minutes. Thus, Alice wants to find a certificate chain that authorizes her to login to H, but has the maximum expiration value among all such certificate chains. A maximally-valid certificate chain is found using the normal operation with the difference that the constraint for the system is kept such that the lower range is $T_{current}$ and the upper range is $\max(V(c_i))$ where $1 \leq i \leq k$ where k is the number of certificates in the system. Given the final automata formed by the algorithm Algorithm- 1, graph search is used to find the path with maximal validity.

5.4.3 Most-recent certificate chain

Let $R(c)$ be the time (relative to the current time) when the cert c was issued or an on-line check was performed on cert c, i.e., $T_{current} - R(c)$ is the actual time of issue of the last on-line check. We call $R(c)$ the recency associated with cert c. The recency of a certificate chain $c_k \circ c_{k-1} \circ \cdots \circ c_1$ is equal to $max_{i-1}^{k} R(c_i)$. Suppose that Aline wants to login to host H. For risk reduction purpose, host H might mandate the use of certificate chain whose recency is no more than ten minutes. In this case, Alice wishes to find a certificate chain that authorizes her to login to H and has the minimum recency among all such chains. Let $c_k \circ c_{k-1} \circ \cdots \circ c_1$ be the certificate chain with minimum recency. If $max_{i=1}^{k} R(c_i)$ is less than or equal to ten minutes, then Alice can use the certificate chain to login to H. Checking recency is straight forward operation in the CAP framework. The temporal range of the certificate is

assigned such that for all the certificates lower range is set as some constant (say 1) and higher range is set as $R(c_i)$. The constraint of the system is set to be such that the lower range of the constraint is the lower range set for all the certificates and the higher range is the recency R required. Application of Algorithm- 1 provides the certificate chain that satisfies the constraints of recency R, if one is available.

5.4.4 Certificate chain with maximal trust

Assume that each certificate c is assigned a trust level $Tr(c)$ by the issuer of the certificate. Intuitively, $Tr(c)$ denotes the confidence that the issuer of c has in the relationship expressed by the certificate c. The trust level of a certificate chain $c_k \circ c_{k-1} \circ \cdots \circ c_1$ is $\bigotimes_{i=1}^{k} Tr(c_i)$. where $\bigoplus$ is defined in Table - 5.2. Suppose that Alice wants to use server S, but has the maximal trust level among all such chains. If such a certificate chain has a trust level above v, Alice can use S.

	D	$\oplus$	$\otimes$	0	1
Validity	$\mathbb{N} \cup \{\pm\infty\}$	max	min	-∞	$+\infty$
Recency	$\mathbb{N} \cup \{\infty\}$	min	max	∞	0
Trust	$\{N, L, M, H\}$	$\sqcap$	$\sqcup$	N	H

Table 5.2: Semiring for Validity, Recency and Trust

Certificate chain with maximal trust is identified using the same method as is employed for identifying privacy-preserving certificate chains. The trust levels are modeled using non-overlapping intervals. The constraint is set so that the certificate chain is maximal with respect to the trust level. For example the trust levels are mapped to non-overlapping intervals in abstract domain as follows: L *(low)* = [5, 10], N *(normal)* = [11, 16], M *(medium)* = [17, 22], H *(high)* = [23, 28].

5.4.5 Order of Events

In additional to temporal constraints which is represented as interval in abstract domain it also possible to check for patterns which are presented as regular language. To manage access control based on execution history [48] it is essential that a facility for modeling and evaluating history be available. The order of occurrence of events is incorporated into the definition of constraint it results in a facility for managing execution of history. The ordering of events define a sequence of occurrence of events which can be used to constraint the access permission. For example, an access control permission may state that the next resource requested can be granted only if the previous access request has been in a particular sequence. The access requests in the past is modeled as events and the order in which these events have happened defines the access control permission.

Meenakshi and Arul Ganesh et al. in [49] describe a framework for decentralized physical access control which allows definition of order of events using WS1S formula. This framework demonstrates that the order and sequence of events can be described using a flexibility of WS1S logic and the same

can be efficiently enforced at runtime. In this framework too MONA [19] has been used as a tool for handling the WS1S formula.

In the following it is described how WS1S formula can be used to describe the sequence of events and how the same can be associated as constraints against SPKI/SDSI certificates. The syntax and semantics of modeling events using WS1S is addressed followed by use of the same to define constraints.

5.4.5.1 Syntax of Event Definition using WS1S

The set of events is denoted by Σ. The *atomic formulae* are given by:

- For each event $e \in \Sigma$, we have a predicate $e(x)$ which represents the fact that the label of the event represented by the variable x is e. This is required to when the ordering of events is to be established.
- For first order variables x, y, the predicate $x < y$ represents the fact that the event corresponding to y occurs immediately after the event corresponding to x in a computation of the system. This defines the natural order of the events.
- For first order variables x, y, the predicate $x \leq y$ represents the fact that the event corresponding to x occurs before the event corresponding to y or x refers to the same event as y in a computation of the system.
- For a first order variable x and a second order variable X, the atomic formula $x \in X$ represents the fact that the event corresponding to the variable x belongs the set of events corresponding to X.

Formulae depicting constraints are built from the atomic formulae, using the following connectives:

- Boolean operators representing negation and disjunction are $\neg$ and $\vee$ respectively. The operators $\wedge$ (conjunction), $\Rightarrow$(implies) and $\equiv$ (equivalence) can be derived from $\neg$ and $\vee$.
- The operators $\forall$(for all) and $\exists$(there exists) is used to quantify over first and second order variables.

To summarize, the syntax of the constraint is WS1S tuned to the context of access control and authorization. Each formula built using the above syntax defines a constraint that needs to be enforced.

5.4.5.2 Semantics of Event Definition using WS1S

Semantics of constraints will be defined using words over the alphabet Σ which defines the set of events of interest. Words are finite sequences of events from Σ. Consider a formula ϕ. ϕ is interpreted over a word w as follows: An *interpretation* of first and second order variables is a function I that assigns a letter of Σ to each first order variable and a finite set of letters of Σ to each second order variable. These

letters occur as positions in a word when a formula (constraint) is interpreted over it. For a formula ϕ, let V_ϕ denote the variables that occur free in ϕ, i.e. they are not in the scope of any quantifier in ϕ . Interpretation is then nothing but a function $I : V_\phi \rightarrow \Sigma$.

The notion of when a word w satisfies a formula ϕ, under an interpretation I is given by $w \models_I \phi$ and is defined inductively as follows:

- $w \models_I e(x)$ if and only if $I(x) = e$.
- $w \models_I x \leq y$ if and only if $I(x)$ occurs before $I(y)$ in the word w.
- $w \models_I x \in X$ if and only if $I(x) \in I(X)$.
- $w \models_I \neg\phi$ if and only if it is not the case that $w \models_I \phi$.
- $w \models_I \phi_1 \vee \phi_2$ if and only if $w \models_I \phi_1$ or $w \models_I \phi_2$.
- $w \models_I (\exists x)\phi$ if and only if there exists an interpretation function I' that extends I by assigning an event to the variable x such that $w \models_{I'} \phi$.
- $w \models_I (\exists X)\phi$ if and only if there exists an interpretation function I' that extends I by assigning a set of events to the variable X such that $w \models_{I'} \phi$.

A *sentence* is a formula without any *free* variables, i.e. all the variables occurring in the formula are bound by a quantifier. Note that sentences can be assigned semantics without any interpretation function. All the constraints will be sentences in WS1S.

5.4.5.3 Example of Event Definition using WS1S

The above description of syntax and semantics of definition of events using WS1S can be explained further using a simple example from the domain of physical access control. A room count event specifies that "user can enter room C only if the number of people in C is less than its capacity". We can express the formula corresponding to this constraint with the following subset of Σ: $\{C_{max}$, C''_{max}, *request_entry_C*, *allow_entry_C*$\}$ in which the letter C_{max} represents the fact that the occupancy of the room C has reached the maximum capacity, the event C''_{max} is the dual of the C_{max} event and represents the fact that the current occupancy of the room C is below the maximum capacity, *request_entry_C* is corresponds to the event that the user has requested an entry into room C, finally the event *allow_entry_C* represents the fact that the user is allowed entry into the room C. WS1S formula corresponding to the constraint is given in Figure 5.10. Semantics of the policy can be understood from the words that satisfy it. E.g. it specifically rules out an occurrence of a later C_{max} happening after a C''_{max} and before a *request_entry_C*, if it is to followed by an *allow_entry_C*. Thus, access will not be granted if C_{max} happened to be the least event before *request_entry_C*, as opposed to its dual-event C''_{max}.

$$\forall x, \forall y$$
$$(C''_{max}(x) \wedge (x \leq y) \wedge request\_entry\_C(y) \wedge$$
$$(\neg \exists z((x \leq z) \wedge (z \leq y)) \wedge (C_{max}(z)) \Rightarrow$$
$$\exists w((y < w) \wedge allow\_entry\_C(w)))$$
$$\forall w(allow\_entry\_C(w) \Rightarrow$$
$$\exists x, \exists y$$
$$(C''_{max}(x) \wedge (x \leq y) \wedge request\_entry\_C(y) \wedge$$
$$(\neg \exists z((x \leq z) \wedge (z \leq y)) \wedge (C_{max}(z)) \wedge (y < w)))$$

Figure 5.10: WS1S formula for the example event definition

Chapter 6

Prototype

6.1 Introduction

The concepts described in this report has been implemented in two different tools. The AWPDS is implemented as a tool called SCAT which allows specification of certificates along with temporal interval constraints. To demonstrate the ease with which complex physical access control constraints can be implemented a tool named FACT has been developed that takes policies from user and outputs WS1S based constraints.

These tools have been designed and implemented using freely available tools like MONA and Moped. These tools have been used as they are robust and help reduce the overall development time.

6.1.1 MONA

MONA [18] is a tool that provides an efficient implementation of the decision procedure for WS1S and WS2S. "Efficient" in the context of the tool is defined to be that it is fast enough that it can be used for most practical applications. MONA translates the formula into a finite-state automaton and analyzes the same and provides "valid" or a counter-example. The tool requires the WS1S formula to be written in a specific language which is refer to as MONA whose syntax is simple and easy to use.

A valid program file containing MONA input is provided to the tool which is compiled into an automaton and provides analysis results on the automaton generated. A MONA program consists of a number of declarations and formulas. An assignment like $[P \longmapsto \{0,1,2,4\}], [Q \longmapsto \{5\}]$ is represented by 0s and 1s in a string as shown in Figure 6.1 where letters are bit-vectors. Each variable is described by a track along the string. Here, the P-track is 111010 and the Q-track is 000001. The positions in the string correspond to natural numbers: a "1" in a position means that the number is in the set, a "0" that it is not. We can define the language associated to a given MONA program as the set of such finite

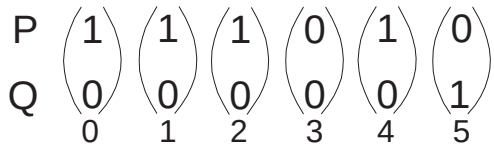

Figure 6.1: The input format for MONA

strings that define satisfying interpretations. For WS1S, this language is regular, which means that it can be recognized by a finite-state automaton. The MONA tool is able to analyze MONA programs given as input to the tool and automatically translating it into the minimum automaton recognizing the set of satisfying interpretations. A common observation is that WS1S can be viewed as generalizations of quantified propositional logic, adding a single "unbounded dimension" orthogonal to the dimension bounded by the number of variables.

MONA has been used in several application. It has been the foundation or successfully integrated into many tools. It is known that arithmetic and logic based approaches are interesting as interesting properties of systems can be encoded.

6.1.2 Moped

The Moped tool from University of Stuttgart, can check a pushdown system, from an initial configuration, against an LTL formula where the atomic predicates consists of a set of atomic symbols that checks the identity of the top stack symbol or the control location. In case the LTL formula is falsified a reduced pushdown system constructed from the original one, that also falsifies the LTL formula, is presented as diagnostic information. In this regard, Moped is a model checker for pushdown systems.

6.2 SPKI/SDSI Certificate Analysis Tool (SCAT)

The CAP framework described have been implemented using MONA tool [19, 20] and WPDS library provided by MOPED [23, 22]. The code base provided by MONA has been converted into a shared library so that the functionality provided by the MONA can be called as a method. The Weighted Pushdown System (WPDS) library provided by MOPED has been extended so that WS1S constraints can be specified against the rules of the WPDS.

This tool named SCAT, for SPKI/SDSI Certificate Analysis tool, allows analysis and reasoning of SPKI/SDSI certificate set augmented with WS1S specification of temporal constraints. Only temporal specification can be provided as input to this tool. The input specification and the tool is such that it can be easily extended so that generic MONA specification can be associated with each certificate. The certificate set and constraints that needs evaluation is provided as input to SCAT using an XML

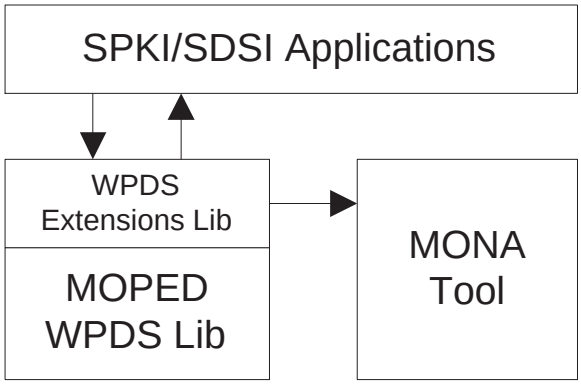

Figure 6.2: SCAT uses MONA and MOPED Library and implements CAP Framework

file. The output provided by SCAT is the certificate chain that meets the system constraints if the constraints can be satisfied given the certificate set and its associated WS1S constraint.

6.3 Facility Access Constraint Tool (FACT)

WS1S logic being a mathematical language makes it cumbersome for an administrator to configure complex policies with. It has also been found that graphical systems are easily understood, and explicitly illustrate the full execution of a system. Properties can be "read off" the diagram, rather than requiring assertions in some symbolic language [50]. Graphical reasoning methods have particular benefits in the context of security protocol and access control system analysis as they allow a more intuitive reading of the specification, making the representation more accessible to the non-specialist.

To get a step closer to intuitive graphical or semi-graphical methods a tool that is more attuned to the domain of physical facility access control has been implemented. The primary objective of this tool named FACT for Facility Access Constraint Tool is to demonstrate that complexity of WS1S can be hidden using domain specific tools. A template based approach for describing policies is implemented by FACT. Details regarding facility topology and events are specified by the administrator in a simple language, which is provided as input to the offline analyzer as shown in Figure 6.3. The template based configuration of policies is done such that it supports *role based access control*, wherein *roles* of users are defined based on the policies that are being enforced on them. *Static* policies as specified using ACLs can also be specified in which case the context of the template becomes empty.

The example below describes how a constraint may be written as a formula in WS1S logic:

for all (*request-for-service*)
if there exists (*relevant-context*) **then**
there exists (*grant-of-service*)

and
if there exists (*grant-of-service*)
there exists (*request-for-service*)
and
there exists (*relevant-context*)

where *request-for-service*, *relevant-context* and *grant-of service* are events of the application. This constraints states that a service is granted if and only if there was a request made for it along with all the conditions required for the service to be granted (context) being true. Depending on the nature of policies, the event relevant-context could also be a formula of WS1S logic. It can include complex features to handle various parameters of dynamism like time, history, context induced by the state of various components of the system.

Examples of policies for two different roles of users for our example facility are given in Table 6.2. Table 6.1 is an example of facility topology as used in Figure 4.7. Different processing stages within

```
# Input topology - List of rooms
rooms :  A,B,C,D,W;
```

```
# Input topology - Neighborhood
neighbor A: C,B,D,W;
neighbor B: A,D;
neighbor C: A,D;
neighbor D: A,B,C;
neighbor W: A;
```

Table 6.1: Facility topology specification

the offline analyzer are shown in Figure 6.3. The *High Level FACDL Constraints Parser* entity is responsible for parsing the high-level description of facility access control constraints using a language called *Facility Access Control Description Language (FACDL)*. The contexts may comprise of two kinds of events, those events that are specific to an individual user by virtue of their own actions, and those events that are observed and constructed by the system for use in one or more user classes. Individual user actions comprise the history of the user. The high level constraints parser parses the specifications of such events declarations and usages, and uses them to construct (predefined) regular expressions with respect to the kind of history in question. For example, the constraint for a regular user for entry to room C, in Table 6.2, requires history event $h2$. *ISSUE ASSET x IN d* implies that user must enter room D, and issue an asset X. Therefore, in order to enter room C, the user history must contain the sequence *request_entry_D*, *allow_entry_D*, *request_asset_X*, *issue_asset_X*. $h2$ is therefore written as a regular expression representing this sequence. Also, policy templates starting with *CAN_ENTER*, depending upon the kind of context that they use, are substituted by corresponding WS1S formulae. For example, the policy of a *regular* user for room C is translated into the WS1S formula of Figure 4.5.

# Define system context EVENT C_{max}: IS count event USES user-entry IN C USES user-exit FROM C PARAM_val GEQ 10 PARAM_user-class EQ regular PARAM_room EQ C EVENT *escort_Timer*: IS timer event USES user-entry IN SELF USES user-exit FROM SELF PARAM_val EQ 10 PARAM_user-class EQ regular EVENT *Regular_escort*: IS timed event USES *escort_Timer* PARAM_escort-class EQ regular PARAM_room EQ SELF	# Role based policy for Regular users policyclass regular: CAN_ENTER W on $\overline{h2}$ CAN_ENTER A CAN_ENTER B ON_CONTEXT $\overline{B_{max}}$ AND $h2$ CAN_ENTER C ON_CONTEXT $\overline{C_{max}}$ CAN_ENTER D ON_CONTEXT $\overline{h1}$ policyclass visitor: CAN_ENTER W CAN_ENTER A ON_CONTEXT *Regular_escort* CAN_ENTER B ON_CONTEXT *Regular_escort* CAN_ENTER C ON_CONTEXT *Regular_escort* # Default deny for room D
# User history based context HISTORY $h1$: ANTI-PASSBACK IN D HISTORY $h2$: ISSUE ASSET X IN D	

Table 6.2: Context and policy specifications

Maintenance of system context, on the other hand, is the responsibility of the system as a whole. Assume that B_{max} is defined similar to C_{max}. It is an example of system context that is true if the count of regular users in room B is greater than or equal to 10. For a regular user to enter room B, B_{max} must be false, i.e. $\overline{B_{max}}$ must be true. Final grant/deny decision, however, is taken after evaluating both the system context and the history $h2$ of the user. The offline analyzer, therefore, must also generate configuration information for the context modeling machinery that dictates how contexts are handled. The *USES* keyword in Table 6.2 describes dependency of a context event on other events. Here it is assumed that application events *user-entry* (to a room) and *user-exit* (from a room) are events that are generated every time a user enters or leaves a room respectively.

The *SELF* keyword is a syntactic sugar. The *Regular_escort* context is defined once, but is used in connection with various rooms. We could have defined this context with respect to individual rooms in which we foresee its use. Alternatively, we use the keyword *SELF* for the *room* parameter, and instantiate this context for rooms A, B, and C at compile-time as and when it is encountered in any user's policy.

The FACDL is a proprietary language that has been defined with focus on the requirements for physical access control. The High Level FACDL Constraints parser is a compiler that checks for the

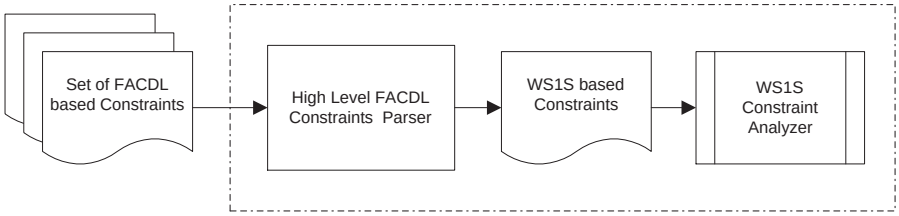

Figure 6.3: Organization of FACT

syntax and the semantics of the constraint defined using FACDL. The parser also generates WS1S based constraints for valid FACDL input. The WS1S constraints analyzer uses MONA to analyze the WS1S constraints generated.

Chapter 7

Summary, Conclusions and Future Work

The WPDS formalism and SPKI/SDSI system is known to have direct correlation between them. Given a SPKI/SDSI certificate set it is possible to define an equivalent WPDS in which the keys of certificate set becomes the control locations and the identifiers form the stack alphabet. The set of labeled rewrite rule becomes the transitions of WPDS. This equivalence allows the user of WPDS reachability algorithm to answer the SPKI/SDSI authorization questions. WPDS allows association of weights to each rule which is used to constrain the search when solving Generalized Predecessor Problem. A more generalized constraint can be defined by associating validity of a rule to the membership to a regular language. In the framework described here WS1S is used as a representation for the regular language of interest as the set of all valid interpretations of a WS1S formula can be represented by a finite state automata. This framework coupled with WPDS gives rise to an Augmented WPDS (AWPDS) which allows evaluation of SPKI/SDSI certificates which are associated to complex constraints and is powerful enough to be able to specify wide variety of constraints which include temporal, spatial and other constraints defined in the literature. The framework called Comprehensive Authorization Problem (CAP) based on WPDS library and MONA tools. This tool named SCAT, for SPKI/SDSI Certificate Analysis tool, allows analysis and reasoning of SPKI/SDSI certificate set augmented with WS1S specification of temporal constraints. In addition to SCAT another tool/parser-compiler has been implemented to demonstrate that WS1S can be used even for the definition of constraints from the physical access control domain. This tool named FACT for Facility Access Constraint Tool provides a domain specific language for the definition of access control constraints typically used for physical access control.

In addition to the work presented here there many improvements and extensions that are possible which are listed below;

- SCAT can be extended in the following ways:
  - Making constraints specification more generic so that flexible constraints can be specified against certificates. This requires changing the input format so that comprehensive WS1S formula, expressed in MONA language, can be associated against each certificate.
  - Confirming to standards like XACML [51], which is a markup language that has been approved by OASIS. XACML promises to standardize policy management and access decision. It defines a general policy language used to protect resources as well as an access decision language. Conformance to XACML will provide interoperability to existing systems and provide an opportunity for deployment in real world scenario.
  - Integration to available SPKI/SDSI implementations. Many implementation of SPKI/SDSI 2.0 standards are available (e.g. Pisces, JSDSI, SPKI/SDSI certificate infrastructure). Extension of these libraries and systems will demonstrate the power of CAP framework proposed.
- Additional security framework proposed in the literature can be modeled using the CAP framework. This requires further investigation. This thesis has only taken few example from the literature. To demonstrate both the power and limitations of the approach discussed the framework should be used on other examples and access control scenarios.
- Optimization of the tool by tighter integration with MONA source code is required. Such an optimization will make the tool usable in real-world scenario.
- A distributed version of the CAP framework need to be developed so that the case of distributed certificate chain discovery, as handled in [16], can be demonstrated.

Appendix A

Example Space Constraints Using WS1S

A.1 Overview

The domain of physical access control deals with policies that deny or grant access to specific physical spaces in a facility. The physical space of interest in a facility which is a room. The access to a room is granted through a door(s). To gain entry into a room the user makes a request for entry into the room. This request results in either a 'allow' or 'deny' action based on the policies defined for the user. The policies defined for the user can be static- which means the access decision are taken only based on parameter(s) that do not mutate or change dynamically, or dynamic - which means that policy is based on parameters or attributes that varies with time or context in which the request is made.

In what follows constraints are defined using MONA, which is an language for specifying WS1S formula, for a simple facility that has four rooms - R1, R2, R3 and R4. A set of events are defined to illustrate how events can be combined together for specifying constraints. The entire code that specifies the constraints is organized into 'decl.mona', 'atomic.mona', 'lib.mona' and 'example.mona' which are described in separate sections.

A.1.1 Declaration

The MONA file 'decl.mona' contains the set of common declarations that are used in other files.

```
# The statement below states that the elements of $
is such that it contains successor elements.var2 $
where ~ex1 p where true: p notin $ & p+1 in $;
allpos $;

# The declarations restrict all variables to $.
defaultwhere1(p) = all1 r: r < p => r in $;
defaultwhere2(P) = all1 p: p in P => all1 r:
r < p => r in $;

# declare a string of 8-bit vectors
var2 bit0 where bit0 sub $,
     bit1 where bit1 sub $,
     bit2 where bit2 sub $,
     bit3 where bit3 sub $,
     bit4 where bit4 sub $,
     bit5 where bit5 sub $,
```

```
    bit6 where bit6 sub $,
    bit7 where bit7 sub $;
```

A.1.2 Library Code

The common set of predicates that are used elsewhere is grouped together into the file 'lib.mona'. The file contains predicates that evaluate if an input sequence defines presence is a room, or if a specific event has occurred as seen from the input sequence given. A predicate defined is related to anti-passback. In the context of physical facility access control anti-passback specifies the constraints that a user should be denied entry into a room if valid exit from the room is not seen. This constraint avoid misuse of authorization provided to a user.

```
# Lib predicates

pred in_R2(var1 p, var1 q) =
ex1 r,s,s':
request_into_R2(r,s) & p <= r
&
allow_into_R2(s,s') & s' <= q
&
~ex1 t,t': allow_into_R3(t,t') & s < t & t' <= q
|
allow_into_R1(t,t') & s < t & t' <= q
;

pred antipassback_violation_R2(var1 p, var1 q) =
p < q
&
ex1 r,s,s',u,u':
request_into_R2(r,s) & p <= r
&
allow_into_R2(s,s') & s' <= q
&
(
~ex1 t,t': allow_into_R3(t,t') & s < t & t' <= q
|
allow_into_R1(t,t') & s < t & t' <= q
)
&
request_into_R2(u,u') & s < u & u' <= q
;

pred is_last_Za(var1 p, var1 q) =
p < q
&
ex1 t,t':
is_Za(t,t') & p <= t & t' <= q
&
(
~ex1 v,v': is_Za'(v,v') & t < v & v' <= q
);

pred is_last_Za'(var1 p, var1 q) =
```

```
p < q
&
ex1 t,t':
is_Za'(t,t') & p <= t & t' <= q
&
(
~ex1 v,v': is_Za(v,v') & t < v & v' <= q
);

pred is_last_Zb(var1 p, var1 q) =
p < q
&
ex1 t,t':
is_Zb(t,t') & p <= t & t' <= q
&
(
~ex1 v,v': is_Zb'(v,v') & t < v & v' <= q
);

pred is_last_Zb'(var1 p, var1 q) =
p < q
&
ex1 t,t':
is_Zb'(t,t') & p <= t & t' <= q
&
(
~ex1 v,v': is_Zb(v,v') & t < v & v' <= q
);
```

A.1.3 Atomic Formulae

A set of atomic predicates that are used during the formulation of higher level of constraints is defined in 'atomic.mona' file. A binary code is allocated to each user-generated and system-generated events so that the occurrence of the events is a string of input can be identified.

```
# String identifiers of all events.

# +---------------------------------------+

#Rooms: R1, R2, R3, R4
#events_on_rooms : R, A, D
#other events: Za, Zb
#
#----Generated events-------
#
#R_R1 | 0 | 0000 |
#A_R1 | 1 | 0001 |
#D_R1 | 2 | 0010 |
#R_R2 | 3 | 0011 |
#A_R2 | 4 | 0100 |
#D_R2 | 5 | 0101 |
#R_R3 | 6 | 0110 |
```

```
#A_R3 | 7 | 0111 |
#D_R3 | 8 | 1000 |
#R_R4 | 9 | 1001 |
#A_R4 | 10 | 1010 |
#D_R4 | 11 | 1011 |
#Za | 12 | 1100 |
#Za'| 13 | 1101 |
#Zb | 14 | 1110 |
#Zb'| 15 | 1111 |

# +-----------------------------------------+

# Room R1

pred request_into_R1(var1 p, var1 q) =
q = p + 1 &
p notin bit0 & p notin bit1 &
p notin bit2 & p notin bit3;
pred allow_into_R1(var1 p, var1 q) =
q = p + 1 &
p in bit0 & p notin bit1 &
p notin bit2 & p notin bit3;
pred deny_into_R1(var1 p, var1 q) =
q = p + 1 &
p notin bit0 & p in bit1 &
p notin bit2 & p notin bit3;

# Room R2

pred request_into_R2(var1 p, var1 q) =
q = p + 1 &
p in bit0 & p in bit1 &
p notin bit2 & p notin bit3;
pred allow_into_R2(var1 p, var1 q) =
q = p + 1 &
p notin bit0 & p notin bit1 &
p in bit2 & p notin bit3;
pred deny_into_R2(var1 p, var1 q) =
q = p + 1 &
p in bit0 & p notin bit1 &
p in bit2 & p notin bit3;

# Room R3

pred request_into_R3(var1 p, var1 q) =
q = p + 1 &
p notin bit0 & p in bit1 &
p in bit2 & p notin bit3;
pred allow_into_R3(var1 p, var1 q) =
q = p + 1 &
p in bit0 & p in bit1 &
p in bit2 & p notin bit3;
pred deny_into_R3(var1 p, var1 q) =
```

```
q = p + 1 &
p notin bit0 & p notin bit1 &
p notin bit2 & p in bit3;


# Za.
pred is_Za(var1 p, var1 q) =
q = p + 1 &
p notin bit0 & p notin bit1 &
p in bit2 & p in bit3;

# Za'.
pred is_Za'(var1 p, var1 q) =
q = p + 1 &
p in bit0 & p notin bit1 &
p in bit2 & p in bit3;

# Zb.
pred is_Zb(var1 p, var1 q) =
q = p + 1 &
p notin bit0 & p in bit1 &
p in bit2 & p in bit3;

# Zb'.
pred is_Zb'(var1 p, var1 q) =
q = p + 1 &
p in bit0 & p in bit1 &
p in bit2 & p in bit3;
```

A.1.4 Example Policy Set

The actual set of constraints that are associated to a group of user is given here. The set of users are grouped into classes named U1, U2, U3, U4 and Uadm. The Uadm class is the administrative user who has special permissions. The other classes of users are arbitrary and may be used defined. In the constraints defined below the dual of an event, say Z, is presented with a apostrophe (') as in Z'.

```
# Template predicates

include "decl.mona";
include "atomic.mona";
include "lib.mona";

# ------ Policyclass U1 -------------

pred can_enter_R1_on_Za'(var1 p, var1 q) =
all1 r,s:
request_into_R1(r,s) & p <= r
&
is_last_Za'(p,r)
=>
ex1 s': allow_into_R1(s,s') & s' <= q;

pred cannot_enter_R1_on_Za(var1 p, var1 q) =
```

```
all1 r,s:
request_into_R1(r,s) & p <= r
&
is_last_Za(p,r)
=>
ex1 s': deny_into_R1(s,s') & s' <= q;

pred can_enter_R3(var1 p, var1 q) =
all1 r,s: request_into_R3(r,s) & p <= r => ex1 s'
: allow_into_R3(s,s') & s' <= q;


pred policyclass_U1(var1 p, var1 q)=
can_enter_R1_on_Za'(p,q)
&
cannot_enter_R1_on_Za(p, q)
&
can_enter_R3(p,q)
;



# ------ Policyclass U2 -------------

pred can_enter_R2_on_Zb_and_Za'(var1 p, var1 q) =
all1 r,s:
request_into_R2(r,s) & p < r
&
(
is_last_Zb(p,r)
&
is_last_Za'(p,r)
)
=>
ex1 s': allow_into_R2(s,s') & s' <= q;

pred cannot_enter_R2_on_Zb'_or_Za(var1 p, var1 q) =
all1 r,s:
request_into_R2(r,s) & p < r
&
(
is_last_Zb'(p,r)
|
is_last_Za(p,r)
)
=>
ex1 s': deny_into_R2(s,s') & s' <= q;

pred can_enter_R3_on_Zb_and_Za'(var1 p, var1 q) =
all1 r,s:
request_into_R3(r,s) & p < r
&
(
```

```
is_last_Zb(p,r)
&
is_last_Za'(p,r)
)
=>
ex1 s': allow_into_R3(s,s') & s' <= q;

pred cannot_enter_R3_on_Zb'_or_Za(var1 p, var1 q) =
all1 r,s:
request_into_R3(r,s) & p < r
&
(
is_last_Zb'(p,r)
|
is_last_Za(p,r)
)
=>
ex1 s': deny_into_R3(s,s') & s' <= q;

# cannot_enter_R1_on_Za
# can_enter_R1_on Za'

pred policyclass_U2(var1 p, var1 q) =
can_enter_R2_on_Zb_and_Za'(p, q)
&
cannot_enter_R2_on_Zb'_or_Za(p, q)
&
can_enter_R3_on_Zb_and_Za'(p, q)
&
cannot_enter_R3_on_Zb'_or_Za(p, q)
&
cannot_enter_R1_on_Za(p,q)
&
can_enter_R1_on_Za'(p,q)
;

# ------ Policyclass U3 -------------

pred can_enter_R1(var1 p, var1 q) =
all1 r,s: request_into_R1(r,s) & p <= r => ex1 s'
: allow_into_R1(s,s') & s' <= q;

pred can_enter_R2(var1 p, var1 q) =
all1 r,s: request_into_R2(r,s) & p <= r => ex1 s'
: allow_into_R2(s,s') & s' <= q;

# can_enter_R3

pred policyclass_U3(var1 p, var1 q) =
can_enter_R1(p, q)
&
can_enter_R2(p, q)
&
```

```
can_enter_R3(p,q)
;

# ------ Policyclass U4 -------------


pred can_enter_R2_on_Zb_and_not_Za(var1 p, var1 q) =
all1 r,s:
request_into_R2(r,s) & p < r
&
(
is_last_Zb(p,r)
&
~is_last_Za(p,r)
)
=>
ex1 s': allow_into_R2(s,s') & s' <= q;

pred cannot_enter_R2_on_Zb'_or_not_Za'(var1 p, var1 q) =
all1 r,s:
request_into_R2(r,s) & p < r
&
(
is_last_Zb'(p,r)
|
~is_last_Za'(p,r)
)
=>
ex1 s': deny_into_R2(s,s') & s' <= q;

pred cannot_enter_R2_on_antipassback_violation_R2(var1 p,
var1 q) =
all1 r,s,t,u:
request_into_R2(r,s) & p <= r
&
p <= t & t <= r
&
antipassback_violation_R2(t,s)
=>
ex1 s': deny_into_R2(s,s') & s' <= q;


pred can_enter_R2_on_not_antipassback_violation_R2(var1 p,
var1 q) =
all1 r,s,t:
request_into_R2(r,s) & p <= r
&
p <= t & t <= r
&
~antipassback_violation_R2(t,s)
=>
ex1 s': allow_into_R2(s,s') & s' <= q;
```

```
pred can_enter_R3_on_Zb_and_not_Za(var1 p, var1 q) =
all1 r,s:
request_into_R3(r,s) & p < r
&
(
is_last_Zb(p,r)
&
~is_last_Za(p,r)
)
=>
ex1 s': allow_into_R3(s,s') & s' <= q;

pred cannot_enter_R3_on_Zb'_or_not_Za'(var1 p,
var1 q) =
all1 r,s:
request_into_R3(r,s) & p < r
&
(
is_last_Zb'(p,r)
|
~is_last_Za'(p,r)
)
=>
ex1 s': deny_into_R3(s,s') & s' <= q;

# cannot_enter_R1_on_Za
# can_enter_R1_on_Za'

pred policyclass_U4(var1 p, var1 q) =
can_enter_R2_on_Zb_and_not_Za(p, q)
&
cannot_enter_R2_on_Zb'_or_not_Za'(p, q)
&
cannot_enter_R2_on_antipassback_violation_R2(p, q)
&
can_enter_R2_on_not_antipassback_violation_R2(p, q)
&
can_enter_R3_on_Zb_and_not_Za(p, q)
&
cannot_enter_R3_on_Zb'_or_not_Za'(p, q)
&
cannot_enter_R1_on_Za(p,q)
&
can_enter_R1_on_Za'(p,q)
;
# ------ Policyclass Uadm -------------

# Note : This is an expand - not a default restrict

# can_enter_R1
# can_enter_R2
# can_enter_R3
```

```
pred policyclass_Uadm(var1 p, var1 q) =
can_enter_R1(p, q)
&
can_enter_R2(p, q)
&
can_enter_R3(p, q)
;


# +-----------------------------------------+
# Random other policies....
pred cannot_enter_R1(var1 p, var1 q) =
all1 r,s: request_into_R1(r,s) & p <= r => ex1 s'
: deny_into_R1(s,s') & s' <= q;

pred cannot_enter_R2(var1 p, var1 q) =
all1 r,s: request_into_R2(r,s) & p <= r => ex1 s'
: deny_into_R2(s,s') & s' <= q;

pred cannot_enter_R3(var1 p, var1 q) =
all1 r,s: request_into_R3(r,s) & p <= r => ex1 s'
: deny_into_R3(s,s') & s' <= q;

pred can_enter_R2_on_Za(var1 p, var1 q) =
all1 r,s:
request_into_R2(r,s) & p <= r
&
is_last_Za(p,r)
=>
ex1 s': allow_into_R2(s,s') & s' <= q;

pred cannot_enter_R2_on_Za'(var1 p, var1 q) =
all1 r,s:
request_into_R2(r,s) & p <= r
&
is_last_Za'(p,r)
=>
ex1 s': deny_into_R2(s,s') & s' <= q;

pred can_enter_R2_on_Za_and_Zb(var1 p, var1 q) =
all1 r,s:
request_into_R2(r,s) & p < r
&
(
is_last_Za(p,r)
&
is_last_Zb(p,r)
)
=>
ex1 s': allow_into_R2(s,s') & s' <= q;

pred can_enter_R2_on_Za_or_Zb(var1 p, var1 q) =
all1 r,s:
```

```
request_into_R2(r,s) & p < r
&
(
is_last_Za(p,r)
|
is_last_Zb(p,r)
)
=>
ex1 s': allow_into_R2(s,s') & s' <= q;
# +-----------------------------------------------+


can_enter_R2(0,max($)+1);
cannot_enter_R2(0,max($)+1);
can_enter_R2(0,max($)+1) &
can_enter_R2_on_Za(0,max($)+1) &
cannot_enter_R2_on_Za'(0,max($)+1);
in_R2(0,max($)+1);
antipassback_violation_R2(0,max($)+1);
cannot_enter_R2_on_antipassback_violation_R2(0,max($)+1);
can_enter_R2_on_not_antipassback_violation_R2(0,max($)+1);
can_enter_R2_on_Za_and_Zb(0,max($)+1);
can_enter_R2_on_Za_or_Zb(0,max($)+1);

policyclass_U1(0,max($)+1)&
policyclass_U2(0,max($)+1)&
policyclass_U3(0,max($)+1)&
policyclass_U4(0,max($)+1)&
policyclass_Uadm(0,max($)+1));

can_enter_R1(0,max($)+1) |
can_enter_R2(0,max($)+1);
can_enter_R3(0,max($)+1);
```

References

[1] T. Y. Woo and S. S. Lam, "Authorization in distributed systems: A formal approach," in *SP '92: Proceedings of the 1992 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 1992, p. 33.

[2] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen, "Spki certificate theory (rfc 2693)," September 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2693.txt

[3] A. J. Lee and M. Winslett, "Safety and consistency in policy-based authorization systems," in *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*. New York, NY, USA: ACM Press, 2006, pp. 124–133.

[4] X. Zhang, M. Nakae, M. J. Covington, and R. Sandhu, "A usage-based authorization framework for collaborative computing systems," in *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*. New York, NY, USA: ACM Press, 2006, pp. 180–189.

[5] M. J. Covington, W. Long, S. Srinivasan, A. K. Dev, M. Ahamad, and G. D. Abowd, "Securing context-aware applications using environment roles," in *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*. New York, NY, USA: ACM Press, 2001, pp. 10–20.

[6] N. Li, B. N. Grosof, and J. Feigenbaum, "Delegation logic: A logic-based approach to distributed authorization," *ACM Trans. Inf. Syst. Secur.*, vol. 6, no. 1, pp. 128–171, 2003.

[7] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati, "An access control model supporting periodicity constraints and temporal reasoning," *ACM Trans. Database Syst.*, vol. 23, no. 3, pp. 231–285, 1998.

[8] E. Bertino, P. Samarati, and S. Jajodia, "Authorizations in relational database management systems," in *CCS '93: Proceedings of the 1st ACM conference on Computer and communications security*. New York, NY, USA: ACM Press, 1993, pp. 130–139.

[9] M. Balasubramanian, A. Bhatnagar, N. Chaturvedi, A. D. Chowdhury, and A. Ganesh, "A framework for decentralized access control," in *2007 ACM Symposium on InformAtion, Computer and Communications Security (ASIACCS'07)*, 2007.

[10] M. Balasubramanian, N. Chaturvedi, A. D. Chowdhury, and A. Ganesh, "A framework for rapid-prototyping of context based ubiquitous computing applications," in *SUTC '06: Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing - Vol 1 (SUTC'06)*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 306–311.

[11] D. K. W. Chiu, C. Wang, H. fung Leung, I. Kafeza, and E. Kafeza, "Supporting the legal identities of contracting agents with an agent authorization platform," in *ICEC '05: Proceedings of the 7th international conference on Electronic commerce*. New York, NY, USA: ACM Press, 2005, pp. 721–728.

[12] G.-C. Roman, C. Julien, and J. Payton, "A formal treatment of context-awareness." in *FASE*, ser. Lecture Notes in Computer Science, M. Wermelinger and T. Margaria, Eds., vol. 2984. Springer, 2004, pp. 12–36.

[13] R. J. Hulsebosch, A. H. Salden, M. S. Bargh, P. W. G. Ebben, and J. Reitsma, "Context sensitive access control," in *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*. New York, NY, USA: ACM Press, 2005, pp. 111–119.

[14] J. R. Büchi, "Weak second order arithmetic and finite automata," *Z. Math. Logik Grundlag. Math.*, vol. 6, pp. 66–92, 1960.

[15] S. Schwoon, S. Jha, T. Reps, and S. Stubblebine, "On generalized authorization problems," *csfw*, vol. 00, p. 202, 2003.

[16] S. Schwoon, H. Wang, S. Jha, and T. Reps, "Distributed certificate-chain discovery in SPKI/SDSI," Computer Sciences Department, University of Wisconsin, Tech. Rep. TR-1526, August 2005.

[17] S. Jha and T. Reps, "Model checking SPKI/SDSI," *Journal of Computer Security*, vol. 12, pp. 317–353, 2004.

[18] N. Klarlund and A. Møller, *MONA Version 1.4 User Manual*, BRICS, Department of Computer Science, University of Aarhus, January 2001, notes Series NS-01-1. Available from `http://www.brics.dk/mona/`. Revision of BRICS NS-98-3.

[19] N. Klarlund, A. Møller, and M. I. Schwartzbach, "MONA implementation secrets," *International Journal of Foundations of Computer Science*, vol. 13, no. 4, pp. 571–586, 2002, world Scientific Publishing Company. Earlier version in Proc. 5th International Conference on Implementation and Application of Automata, CIAA '00, Springer-Verlag LNCS vol. 2088.

[20] J. Elgaard, N. Klarlund, and A. Møller, "MONA 1.x: new techniques for WS1S and WS2S," in *Proc. 10th International Conference on Computer-Aided Verification, CAV '98*, ser. LNCS, vol. 1427. Springer-Verlag, June/July 1998, pp. 516–520.

[21] J. G. Henriksen, J. L. Jensen, M. E. J., N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm, "Mona: Monadic second-order logic in practice," in *TACAS '95: Proceedings of the First International Workshop on Tools and Algorithms for Construction and Analysis of Systems*. London, UK: Springer-Verlag, 1995, pp. 89–110.

[22] S. Schwoon, "Moped: A model-checker for pushdown systems," Computer Sciences Department, University of Wisconsin, Tech. Rep., 2002. [Online]. Available: http://www.fmi.uni-stuttgart.de/szs/tools/moped/

[23] ——, "Wpds: A library for weighted pushdown systems," Computer Sciences Department, University of Wisconsin, Tech. Rep., 2003. [Online]. Available: http://www.fmi.uni-stuttgart.de/szs/tools/wpds/

[24] D. E. Clarke, "Spki/sdsi http server / certificate chain discovery in spki/sdsi," September 2001, supervisor-Frank Thomson Leighton.

[25] D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R. L. Rivest, "Certificate chain discovery in spki/sdsi," *J. Comput. Secur.*, vol. 9, no. 4, pp. 285–322, 2001.

[26] T. Reps, S. Schwoon, S. Jha, and D. Melski, "Weighted pushdown systems and their application to interprocedural dataflow analysis," *Sci. Comput. Program.*, vol. 58, no. 1-2, pp. 206–263, 2005.

[27] A. Bouajjani, J. Esparza, and T. Touili, "A generic approach to the static analysis of concurrent programs with procedures," in *POPL '03: Proceedings of the 30th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. New York, NY, USA: ACM Press, 2003, pp. 62–73.

[28] J. Howell and D. Kotz, "A formal semantics for spki," Hanover, NH, USA, Tech. Rep., 2000.

[29] T. S. Jha, "Analysis of spki/sdsi certificates using model checking," Computer Science Department, University of Wisconsin, 1210 W.Dayton Street, Madison, WI 53706, Tech. Rep., 2001.

[30] M. Abadi, "On sdsi's linked local name spaces," *J. Comput. Secur.*, vol. 6, no. 1-2, pp. 3–21, 1998.

[31] J. Y. Halpern and R. V. der Meyden, "A logical reconstruction of spki," in *CSFW '01: Proceedings of the 14th IEEE Workshop on Computer Security Foundations*. Washington, DC, USA: IEEE Computer Society, 2001, p. 59.

[32] A. W. Appel and E. W. Felten, "Proof-carrying authentication," in *CCS '99: Proceedings of the 6th ACM conference on Computer and communications security*. New York, NY, USA: ACM Press, 1999, pp. 52–62.

[33] F. Pfenning and C. Schürmann, "System description: Twelf - a meta-logical framework for deductive systems," in *CADE-16: Proceedings of the 16th International Conference on Automated Deduction*. London, UK: Springer-Verlag, 1999, pp. 202–206.

[34] N. Li, W. H. Winsborough, and J. C. Mitchell, "Distributed credential chain discovery in trust management," *J. Comput. Secur.*, vol. 11, no. 1, pp. 35–86, 2003.

[35] M. S. Ruben Wolf, Thomas Keinz, "A model for context-dependent access control for web-based services with role-based approach," *Proceedings of the 14th International Workshop on Database and Expert Systems Applications (DEXA'03)*, 2003.

[36] A. C. W. Junzhe Hu, "Dynamic, context-aware access control for distributed healthcare applications," 2003.

[37] D. Gelernter, "Generative communication in linda," *ACM Transactions on Programming Languages and System*, vol. 7, no. 1, pp. 80–112, 1985.

[38] R. Gray, D. Kotz, G. Cybenko, and D. Rus, "D'agents: Security in multiple-language, mobile-agent system," *Mobile Agents and Security*, vol. 1419, pp. 154–187, 1998.

[39] G.-C. R. Christine Julien, Jamie Payton, "Context-sensitive access control for open mobile agent systems," 2004.

[40] D. J. Dougherty, K. Fisler, and S. Krishnamurthi, "Specifying and reasoning about dynamic access-control policies." in *IJCAR*, 2006, pp. 632–646.

[41] F. Afinidad, T. Levin, C. Irvine, T. Nguyen, *et al.*, "Foundation for a Time Interval Access Control Model," *LECTURE NOTES IN COMPUTER SCIENCE*, vol. 3685, p. 406, 2005.

[42] D. Wijesekera and S. Jajodia, "A propositional policy algebra for access control," *ACM Trans. Inf. Syst. Secur.*, vol. 6, no. 2, pp. 286–325, 2003.

[43] J. Esparza, D. Hansel, P. Rossmanith, and S. Schwoon, "Efficient algorithms for model checking pushdown systems," in *CAV '00: Proceedings of the 12th International Conference on Computer Aided Verification*. London, UK: Springer-Verlag, 2000, pp. 232–247.

[44] J. Glasgow, G. Macewen, and P. Panangaden, "A logic for reasoning about security," *ACM Trans. Comput. Syst.*, vol. 10, no. 3, pp. 226–264, 1992.

[45] R. Alur and T. A. Henzinger, "Logics and models of real time: A survey," in *Proceedings of the Real-Time: Theory in Practice, REX Workshop.* London, UK: Springer-Verlag, 1992, pp. 74–106.

[46] J. F. Allen, "Maintaining knowledge about temporal intervals," pp. 361–372, 1990.

[47] A. Bouajjani, J. Esparza, and O. Maler, "Reachability analysis of pushdown automata: Application to model-checking," in *CONCUR '97: Proceedings of the 8th International Conference on Concurrency Theory.* London, UK: Springer-Verlag, 1997, pp. 135–150.

[48] M. Abadi and C. Fournet, "Access control based on execution history," 2003.

[49] M. Balasubramanian, A. Bhatnagar, N. Chaturvedi, A. D. Chowdhury, and A. Ganesh, "A framework for decentralized access control," in *ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, computer and communications security.* New York, NY, USA: ACM Press, 2007, pp. 93–104.

[50] C. Skalka and S. Smith, "Diagrammatic verification of security protocols." [Online]. Available: citeseer.ist.psu.edu/375621.html

[51] T. Moses, "Oasis - extensible access control markup language (xacml) version 2.0," February 2005.

Index